Title

Predicting When Not To Predict

Authors

Brandt, Karl
Long, Darrell DE
Amer, Ahmed

Publication Date

2004

DOI

10.1109/mascot.2004.1348297

Peer reviewed

Predicting When Not To Predict

Karl Brandt and Darrell D. E. Long[†]
University of California, Santa Cruz
Santa Cruz, CA

Ahmed Amer [‡]
University of Pittsburgh
Pittsburgh, PA

Abstract

File prefetching based on previous file access patterns has been shown to be an effective means of reducing file system latency by implicitly loading caches with files that are likely to be needed in the near future. Mistaken prefetching requests can be very costly in terms of added performance overheads including increased latency and bandwidth consumption. Such costs of mispredictions are easily overlooked when considering access prediction algorithms only in terms of their accuracy, but we describe a novel algorithm that uses machine learning to not only improve overall prediction accuracy, but as a means to avoid these costly mispredictions. Our algorithm is fully adaptive to changing workloads, and is fully automated in its ability to refrain from offering predictions when they are likely to be mistaken. Our trace-based simulations show that our algorithm produces prediction accuracies of up to 98%. While this appears to be at the expense of a very slight reduction in cache hit ratios, application of this algorithm actually results in substantial reductions in unnecessary (and costly) I/O operations.

1. Introduction

While both processor and storage hardware have been improving in performance, there has been a growing performance gap, with I/O often being the bottleneck and systems generally having a difficult time keeping up with improvements in their underlying hardware [18, 25]. This has meant that methods to address the I/O performance gap have grown increasingly critical [8]. Traditionally, the basic approach to improving data access performance has been caching in one form or another, and more recently there have been numerous proposals for prefetching algorithms that attempt to improve upon basic caching through the prediction of future data access requests.

The ideal traditional caching scheme would take the form of Belady's MIN algorithm [5], focusing on the decision of what item to remove from the cache. The MIN cache replacement policy simply removed the item that would be needed furthest in the future. Obviously this requires perfect prescience, but does offer a replacement policy that is impossible to improve upon. Even so, improvements are possible if we do not limit ourselves to the question of which object to replace, and broaden our scope to consider what items we can fetch *before an explicit request is actually made*. This has been found to be feasible for I/O access patterns through the observation of past behavior [11]. If successful predictions are made in time, it is possible for the request to find the data already available in the cache. This would effectively eliminate the access time for the storage system, making the data available at main memory speeds.

While there have been numerous studies of data access prediction, one of the few implementation studies found that inaccurate predictions could lead to degradation of system performance in spite of making many accurate predictions [12]. Inaccurate prefetching of objects that will not be used results in the tying up of I/O channels and the waste of valuable cache space. In the absence of I/O preemption, incorrect prefetches will further increase the average latency experienced by cache misses [7]. Consequently, file prefetching algorithms need to be as accurate as possible, providing many accurate predictions, but also minimizing the number of incorrect predictions [3].

The main contribution of this work is to present an effective file and data prefetching algorithm that is both accurate, adaptive, and which automatically learns to avoid making predictions that are unlikely to be correct. Our algorithm uses the machine learning technique of multiple experts and we introduce the idea of a *null* prediction expert to suppress prefetching in instances where the likelihood of an accurate prefetch is low. When looking at cache replacement algorithms, Ari *et al.* [4] found that systems went through periods where different choices of replace-

[†]Supported in part by the National Science Foundation under award CCR-0204358.

[‡]Supported in part by the National Science Foundation under award ANI-0325353.

ment algorithms were best suited to maximizing cache hit ratio. To predict file access patterns we would therefore need algorithms that can adapt to changes in the workloads. Our approach provides an access prediction algorithm that not only adapts to the workload, but learns to avoid making predictions for specific files or data that are unlikely to yield a successful successor prediction.

2. Prediction with Multiple Experts

Multiple experts is a class of machine learning algorithms in which an on-line learner is faced with a series of trials. On each of these trials the learner makes a prediction as to the trial's outcome. Shortly afterward, the true outcome of the trial is revealed in some way. The goal of the learner is to minimize the number of mistakes over all the trials.

The canonical example for multiple expert algorithms is for weather prediction. A pool of meteorologists (experts) are asked to predict tomorrow's weather. The manager (master algorithm) then makes a prediction based on some combination of the set of individual predictions. The following day, the accuracy, or inaccuracy, of the individual meteorologists are judged. After each day (trial), the confidence in the meteorologists is adjusted. The master algorithm can then combine new predictions based on the confidence ratings of the meteorologists. After enough trials, the master algorithm begins to listen mostly to the meteorologists who have predicted well, and largely ignores the experts who have not. Eventually, the master algorithm should be able to predict the weather at least as well as the best meteorologist and possibly better if it can aggregate the predictions of several good meteorologists.

2.1. Weighted Majority Learning

The class of *Weighted Majority* algorithms use the predictions of a number of experts that can be simple static predictions, heuristics, or possibly other machine learning algorithms [16, 23]. Most commonly, the experts are low cost algorithms for making predictions. The master algorithm has no *a priori* knowledge about which experts will perform well over a series of trials.

Each expert is assigned a weight that represents the master algorithm's confidence in that expert's predictions. The master algorithm then uses these weights, in combination with the individual expert's predictions, to determine its own prediction. The simplest combination option is to have the master algorithm predict the same as the expert in which it has the most confidence. This is a simple matter of picking the expert with the current largest weight. Another option is to aggregate the predictions in some way. For example, when attempting to predict a binary value, the master algorithm can sum the product of expert predictions and their weights: $\sum_{i=1}^{N} \omega_i \hat{y}_i$ where ω_i is the weight assigned to expert i, $\hat{y}_i$ is the outcome predicted by expert i, and N is the number of experts. Sums above some threshold result in a master prediction of 1, and sums below the threshold result in 0. This is where the name Weighted Majority comes from. The experts vote on the prediction with the number of votes allocated to each expert based on its weight.

At the end of each trial (*e.g.* after each prediction), the weights of the experts are adjusted based on their performance. *Loss* of expert i on trial t, as shown in Equation 1, is the means of quantifying the difference between the predicted outcome $\hat{y}_{i,t}$ and the actual outcome y_t.

$$L(i,t) = |\hat{y}_{i,t} - y_t| \tag{1}$$

The individual expert weights are updated according to Equation 2. Experts that predict correctly, and consequently have no loss, have their weights unchanged except for a normalization factor. Experts that incur loss, have their weights decreased by some parameter $\beta < 1.0$ raised to the power of the loss function. The normalization is not strictly necessary for the algorithm to work properly. In practice, it is done to avoid arithmetic underflows that would eventually happen from repeatedly decreasing weights.

$$\omega^m_{t+1,i} = \frac{\omega_{i,t}\beta^{L(i,t)}}{\text{normalization factor}}, \text{ where } \beta \in [0,1] \tag{2}$$

Cesa-Bianchi *et al.* showed tighter mistake bounds by setting $\beta = e^{-\eta}$, where η is a tuning parameter to adjust the rate that weights are updated [6]. Littlestone and Warmuth prove a mistake bound of $O(M \log |N|)$ where M is the number of mistakes made by the best performing expert and N is the number of experts [16].

2.2. File Prediction with Multiple Experts

Our multiple expert implementation aims to predict the immediate successor to each file accessed. Each file has a set of experts associated with it. These *file experts* consist of previously observed successors to the file in question. Each of these file experts advocates that one of the previous successor files is likely to be the next successor. Additionally, there is a *null expert* that advocates the best course of action is to make no prediction. Each expert is assigned a weight with the sum of the weights normalized to 1. When a file is accessed, the successor prediction is made based on the expert with the greatest weight. If the *null* expert has the greatest weight, no prediction is issued.

2.2.1. Loss Functions for File Successor Predictions Each file access is a trial of the current file's set of experts. The actual successor reveals the performance of each expert. Experts that predict incorrectly incur loss as a result of their mistakes. The loss function for *file* experts is shown in Equation 3.

$$L(\hat{y}_{i,t}, y_t) = \begin{cases} 0 & \hat{y}_{i,t} = y_t \\ 1 & \hat{y}_{i,t} \neq y_t \end{cases} \quad (3)$$

The *null* expert is only correct when none of the file experts predicted correctly. This requires that the actual successor fail to match any of the file experts. In these cases, the best course of action would have been not to prefetch any file. The loss ρ (where $0 \leq \rho \leq 1$) of not making any prediction when one was possible is some fraction of the cost of an incorrect prediction. A correct *null* prediction still has some loss associated with it when compared with a correct file prediction. Where a misprediction incurs costs, a non-prediction loses the benefit of prefetching. This is not as bad as predicting the wrong file but it still is not as good as having correctly predicted the successor. Consequently, the *null* expert has a modified loss function as shown in Equation 4. We have found that the algorithm was relatively immune to changes in ρ. For our experiments we used $\rho = 0.5$.

$$L(\hat{y}_{\text{null}}, y_t) = \begin{cases} 0 & \text{if } \forall i : \hat{y}_i \neq y_t \\ \rho & \text{otherwise} \end{cases} \quad (4)$$

2.2.2. Determining Predictor Success There is a slight mismatch between how the algorithm determines a trial's outcome and the intuitive system behavior. It is possible for a prefetched object to not match the successor request and yet still be useful. If the prefetched object is used prior to being evicted from the cache, it was beneficial by eliminating a cache miss. Our loss functions could unfairly judge some experts as wrong when in fact they produce a benefit to the system. We chose to accept this more conservative interpretation of success in order to simplify the process of evaluating experts' performance. Otherwise, the outcome of a trial would not be revealed until a prefetched object was evicted from the cache, greatly slowing adaptivity and rendering the evaluation sensitive to specific cache size.

In order to limit the amount of metadata needed by each file, we capped the number of file experts at five. In our experiments, the choice of how many files to track had insignificant effects on our results. This is consistent with prior work that demonstrated how most files have only a few unique successors when observed over periods of several days [3].

3. Experiments and Results

In order to compare our multiple experts algorithm against both traditional passive caching and other prefetching methods, we simulated caches of varying sizes and used prerecorded workload traces. We chose the basic LRU and LFU algorithms, as well as the recent ARC cache replacement algorithm. We also compared against three competitive predictive prefetching schemes applied to an underlying LRU scheme.

3.1. Simulation Methodology

Our primary purpose was to compare the effectiveness of our more conservative multiple experts algorithm against more aggressive prefetching schemes. For this purpose we compared against three previously proposed successor predictors: Last Successor (LS), Stable-Successor (Noah) [3], and group-based prefetching [2]. We measured the performance of the cache replacement policies and compare them to the performance of the predictors employed as cache prefetching algorithms. Unless otherwise stated, all predictors were applied as an enhancement to an underlying LRU cache.

We developed a discrete event simulator that took filtered traces as input and tracked the cache hit ratios and predictive accuracy. While cache hit ratios give an indication of the reduction in I/O latencies they neglect potential costs of prefetching. Incorrect prefetches can cause disk queuing delays that are not reflected in hit ratios. *Prefetching accuracy*, the number of correct prefetches out of the total number of prefetches, is one way of evaluating prefetching performance. The relative importance of cache hit ratios to prefetching accuracy will naturally vary from one system scenario to another, therefore we discuss both metrics. All experiments used traces drawn from three different trace sets:

- **Coda Traces** These traces were collected with the DFSTrace system [17] as part of the Coda project [21]. All system calls on 33 machines from February, 1991 until March of 1993 were recorded. *Barber* was the server with the highest rate of system calls; *Mozart* was a typical single-user workstation.

- **SEER Traces** The SEER [14] traces were system calls taken from single-user work stations used by members of their research group. *Norgay* was traced from September, 1996 to March, 1997. *Erasmus* was traced from September to February, 1997. These two were chosen as typical traces out of the nine collected.

- **HP Traces** These are two traces from *cello*, a server, and *hplajw*, a workstation, collected in 1999

by Hewlett-Packard on systems running HP-UX. Ruemmler and Wilkes [20] traced and analyzed these same machines in 1992. Unlike our other traces, these are block level traces taken at the storage device, though for consistency we refer to data items as "files" throughout this work.

The *hplajw* trace was from February 10, 1999 through February 19, 1999 and contained 2.6 million "file" accesses. Unfortunately, our simulator had trouble running long slices of the *cello* trace and limited us to roughly 80,000 "file" accesses.

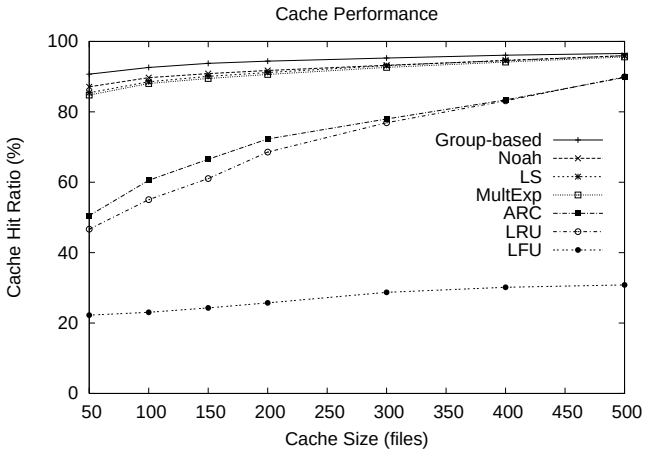
3.2. Cache Hit Ratios

We first looked at cache hit ratios to give some idea of the potential latency reductions of the various prefetching methods. Then we looked at predictive accuracy to give an indication of how much useful work was being done by the prefetchers. In an effort to give a more complete picture of prefetching performance, we looked at the loss incurred by each prefetcher.

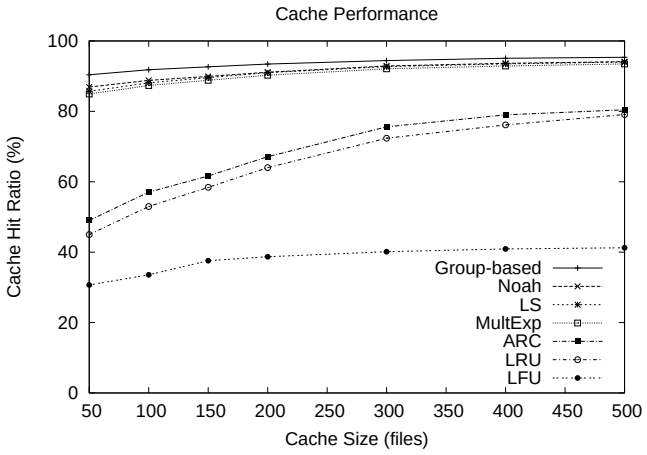
The primary purpose of any prefetching scheme is to increase the cache hit ratios. It is worth distinguishing that we are only interested in the hit ratios for demand fetches, those that normally occur in the request stream. Implicit fetches, that are initiated by prefetching, are unimportant as far as cache hit ratios are concerned.

Figure 1 shows the hit ratios for both our three passive caching algorithms (LRU, LFU and ARC), and four prefetching algorithms (Group-based, Stable Successor, LS, and our proposed Multiple Experts) over a wide range of cache sizes. Each of the prefetching algorithms use LRU as their underlying replacement policy. For most of the traces, the predictive algorithms all outperform the passive caching algorithms. *Barber* (Figure 1(c)) is the one noticeable exception where ARC achieves higher hit ratios for larger cache sizes. Since the choice of replacement algorithms is independent of the prediction algorithm, we reran the simulations with ARC instead of LRU as the underlying replacement algorithm. Figure 2 shows that indeed the prefetching algorithms have higher hit ratios when combined with ARC.

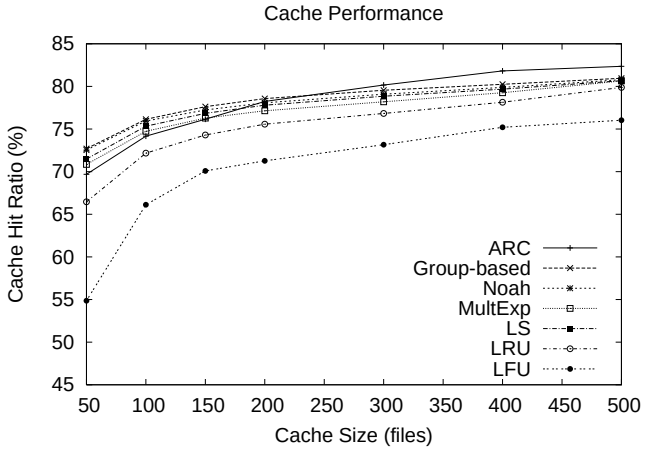
In general, algorithms that issue more prefetches per file access will produce higher hit ratios as long as those predictions have greater likelihood of access than the objects they replace. The results for all traces were similar to those of Figure 1, and confirm that Group-Based prefetching consistently generates higher hit ratios since it issues multiple predictions per file access. Stable Successor (Noah) and Last Successor (LS) achieve slightly lower hit rates since they issue a single prediction per file access with Stable Successor doing better simply because it predicts more accurately than LS. The conservative nature of our proposed



(a) *Norgay* (SEER Workstation)



(b) *Erasmus* (SEER Workstation)



(c) *Barber* (Coda File Server)

Figure 1. Hit Ratios, SEER and Coda Traces

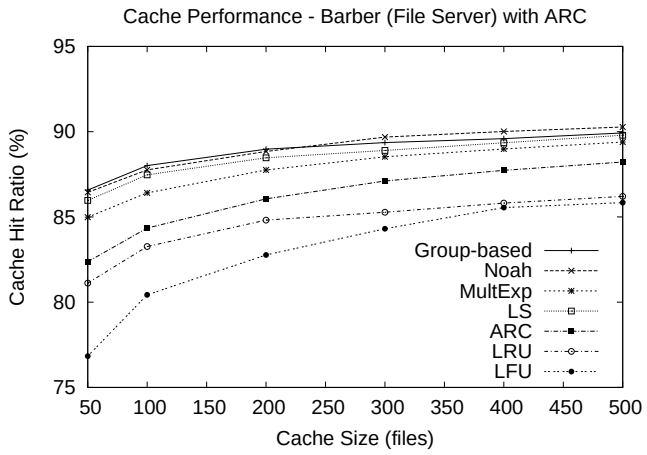


Figure 2. Cache Hit Ratio with ARC

Multiple Experts algorithm lowers its cache hit ratios, as it chooses not to issue predictions when the null expert wins. Nonetheless, it should be noted that Multiple Experts consistently achieves cache hit ratios among those of the highest-performing algorithms tested, even if it does not exceed them.

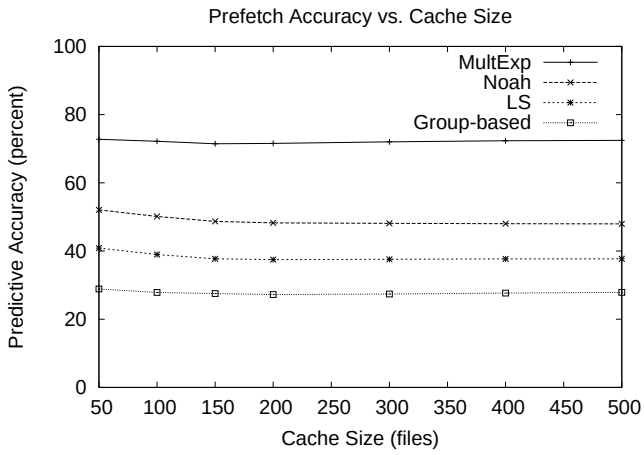
3.3. Prefetch Accuracy

Prefetch accuracy is the number of useful prefetches over the number of prefetches performed. Prefetches are deemed useful if the objects that were prefetched are requested by demand fetches before they are replaced. Even though an algorithm may not accurately predict an immediate successor, its prefetch can still benefit the system as long as the prefetch reduces the demand fetches. Prefetched objects that are never used, lower an algorithm's prefetch accuracy.

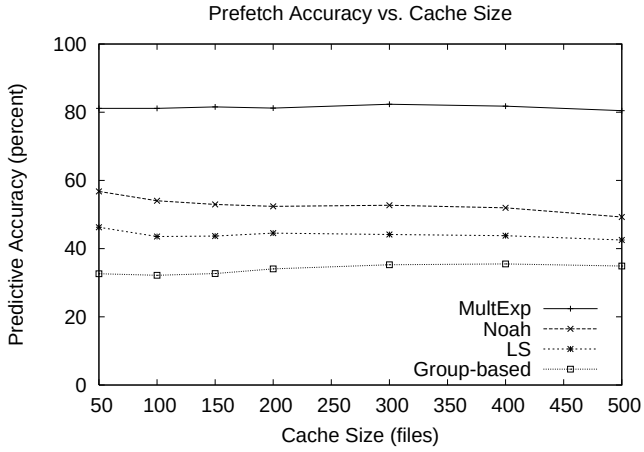
Figures 3 and 4 show that Multiple Experts is able to achieve consistently higher prefetch accuracy than Group-based, Stable Successor or LS prefetching. The null expert is allowing Multiple Experts to refrain from issuing predictions when it is uncertain of being correct. Multiple Experts had a worst observed accuracy of 68% and frequently delivered as high as 98% (the higher values were also true for the SEER traces which are not shown due to space limitations).

3.4. I/O Performance

The primary goal of any caching scheme is to reduce latency and improve the perceived performance of lower-levels in the storage hierarchy. With our simulator we measured the the number I/O requests that are required by the prefetching caches. This is an important measure as it is



(a) *Mozart* (Workstation)



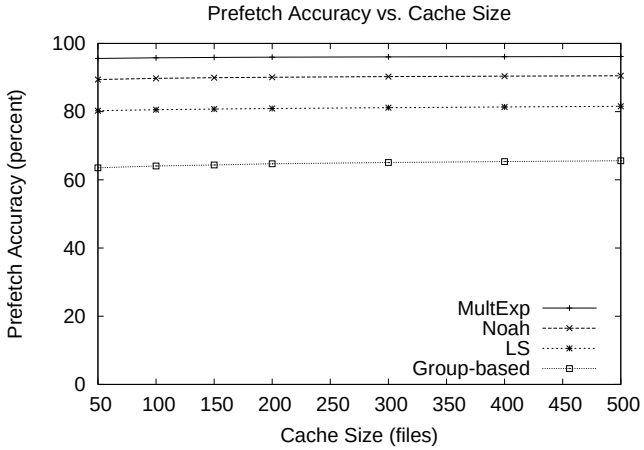
(b) *Barber* (File Server)

Figure 3. Prefetch Accuracy, Coda Traces

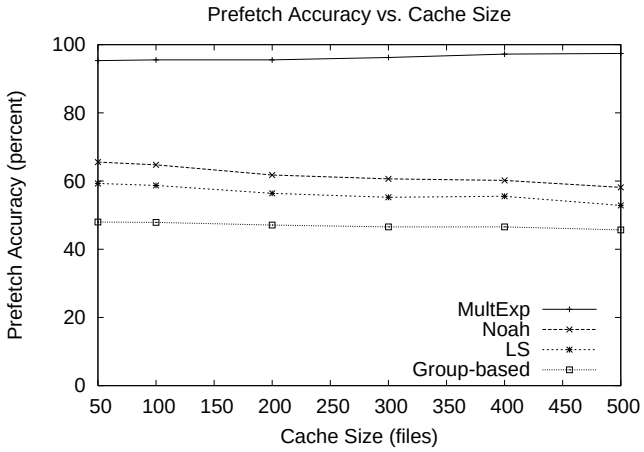
indicative of the time spent by the system waiting for I/O operations to complete.

Figures 5 and 6 show the average number of I/O operations needed per file request for the *Mozart*, *Barber*, *hplajw*, and *Cello* traces. This is a ratio of the number of items retrieved by the cache, compared to the number of requests for items that the cache would have made if it was only performing demand replacment. An cache effective at reducing the I/O workload will have a low ratio. It can be seen from

In all cases our Multiple Experts algorithm requires fewer I/O operations than the Group-based, LS and Noah schemes. This is also true for the SEER traces, which we do not include for space limitations. While Multiple Experts exhibited competitive performance in terms of cache hit rates, it can be seen to dramatically excel in both prefetch accuracy, and I/O reduction. The next scheme in terms of I/O reduction is Noah, also known as Stable Successors [3], which allows for prudence in making predictions,

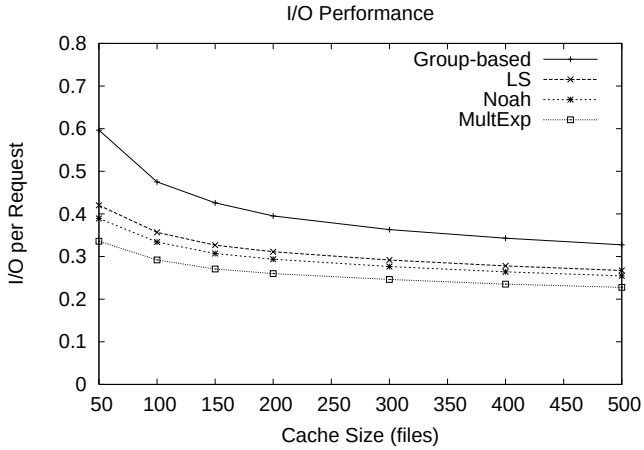


(a) *hplajw* (Workstation)

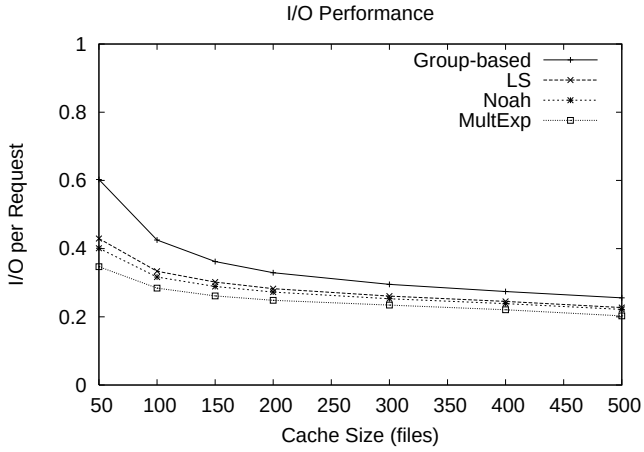


(b) *cello* (Server)

Figure 4. Prefetch Accuracy, HP Traces



(a) *Mozart* (Workstation)



(b) *Barber* (Server)

Figure 5. I/O Performance, Coda Traces

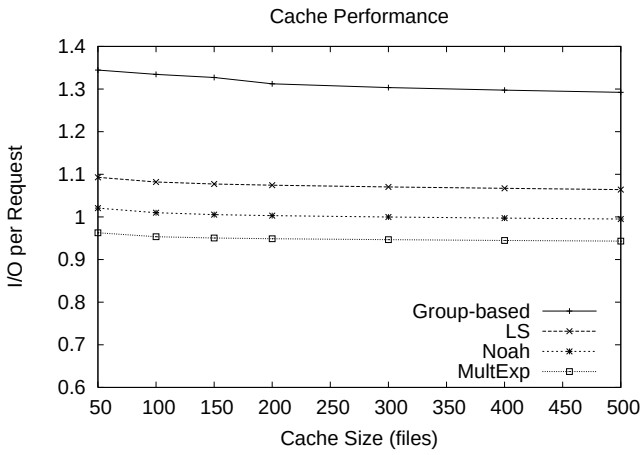
thereby avoiding many unnecessary I/Os. But while Noah uses a global parameter for determining whether or not a prediction is safe, Multiple Experts requires no such parameterization, as it will automatically determine whether the set of candidate successors is promising or not. Furthermore, Multiple Experts performs this evaluation independently for each file, with little more overhead than is required to track the per-file predictive metadata, resulting in the greater accuracy with fewer I/Os.

4. Related Work

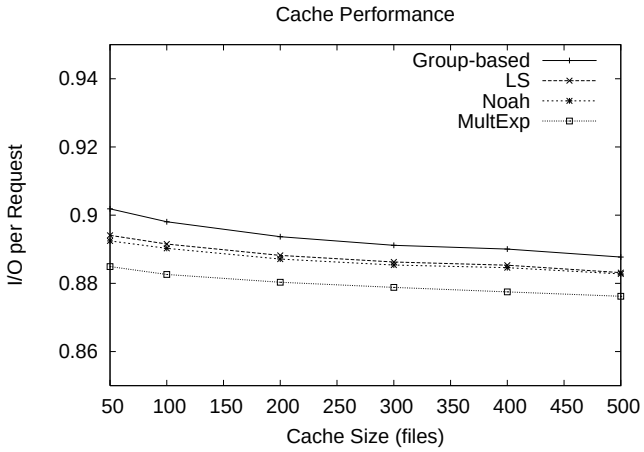
Considerable research has gone into methods of prefetching files. Patterson *et al.* suggested modifying compilers to allow programmers to provide the operating system with hints about future file use [19]. These hints would then be used to prefetch files as the operating system saw fit. Obviously this would require the recompilation of programs to be effective. Additionally, it requires that programmers have accurate knowledge about what files a program will be accessing. Trends in object oriented programming toward more encapsulation and layers of abstraction make this problem more difficult for the programmer.

Kuenning *et al.* built SEER, a prototype file hoarding system for disconnected operation. They introduced the notion of *semantic distance*, which is derived from the order and overlap of file references [14]. A graph is built and used to cluster files likely to be accessed in close proximity. The clustering algorithm was augmented by a series of heuristics tuned to the peculiarities of Unix file access patterns, and in later work discovered that the heuristics were responsible for the improved hoard performance and not the actual clustering methods [13]. The heuristics combined with a simple LRU policy for hoard file replacement were shown to outperform their more complicated clustering scheme.

Griffioen and Appleton proposed automatic file prefetching based on the prior access stream and a rela-



(a) *hplajw* (Workstation)



(b) *Cello* (Server)

Figure 6. I/O Performance, HP Traces

tionship graph [9]. They maintained a directed probability graph for the successors to each file that were accessed within a window, and a successor was prefetched if its estimated probability was above some system threshold value.

Lei and Duchamp maintain a database of access trees for different program executions [15]. When a program is run, the tree database is searched for patterns that match the current accesses. If one is found to match close enough, the remaining files in the tree are prefetched. Yeh *et al.* correlated successor predictions to both programs and specific users [26], masking the more random effects of operating system behavior like context switches.

Vitter and Krishnan [22] first proposed the use of data compression techniques to file prediction, while Kroeger and Long [10] produced the space-efficient and more adaptive *Partitioned Context Model* (PCM). Kroeger and Long went on to develop a Linux prototype implementation which demonstrated that simply making good predictions were not enough to achieve performance improvements, and their solution was to produce predictions further ahead using the *Extended Partitioned Context Model* (EPCM) [12].

Last Successor (LS) is a simple baseline predictor. It predicts the successor to the current access of a file, to be the same as the previous time the file was accessed. This naive heuristic performs surprisingly well and serves as a good starting point for comparing successor predictors. Amer and Long proposed *Noah* [1], which is one of several variations on LS. They later went on to propose mechanisms for assembling groups of successors to a given file. These are targeted toward client-server environments. Servers can aggregate the knowledge gained by observing multiple client file streams in order to predict likely groups of immediate and transitive successors to files [2]. Later work also used multiple heuristics to predict file successors [24]. These included the use of successor stability [3], and more general per-file successor popularity schemes to limit false predictions.

5. Conclusions and Future Work

Good cache management is critical in reducing I/O latency, and access prediction has the potential to eliminate I/O latency or to seriously harm system performance if predictions are mishandled. We have demonstrated an access prediction algorithm that exhibits high prediction accuracy, and improves cache hit rates, whilst simultaneously reducing the likelihood of unnecessary I/O requests and their associated costs. Our Multiple Experts scheme satisfies these apparently mutually exclusive goals, while doing so in an algorithm that automatically learns when not to attempt a prediction, and does so successfully on a per-file basis with no critical parameters that would require user setting or intervention.

References

[1] A. Amer and D. D. E. Long. Noah: Low–cost file access prediction through pairs. In *Proceedings of the 20th IEEE International Performance, Computing and Communications Conference (IPCCC '01)*, pages 27–33. IEEE, Apr. 2001.

[2] A. Amer, D. D. E. Long, and R. C. Burns. Group–based management of distributed file caches. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS '02)*, Vienna, Austria, July 2002. IEEE.

[3] A. Amer, D. D. E. Long, J.-F. Pâris, and R. C. Burns. File access prediction with adjustable accuracy. In *Proceedings of the International Performance Conference on Computers and Communication (IPCCC '02)*, Phoenix, Apr. 2002. IEEE.

[4] I. Ari, A. Amer, R. Gramacy, E. L. Miller, S. A. Brandt, and D. D. E. Long. ACME: Adaptive Caching Using Multiple Experts. In *Proceedings in Informatics*, volume 14, pages 143–158. Carleton Scientific, 2002.

[5] L. A. Belady and F. P. Palermo. On–line measurement of paging behavior by the multivalued MIN algorithm. *IBM Journal of Research and Development*, 18(1):2–19, Jan. 1974.

[6] N. Cesa-Bianchi, Y. Freund, D. Haussler, D. P. Helmbold, R. E. Schapire, and M. K. Warmuth. How to use expert advice. *Journal of the Association for Computing Machinery*, 44(3):427–485, May 1997.

[7] Z. Dimitrijevic, R. Rangaswami, and E. Chang. Design and Implementation of Semi-preemptible IO. In *Proceedings of the 2003 Conference on File and Storage Technologies (FAST)*, San Francisco, CA, Mar. 2003. USENIX.

[8] J. Gray and P. Shenoy. Rules of thumb in data engineering. In *Proceedings of the 16th International Conference on Data Engineering (ICDE '00)*, pages 3–12, San Diego, CA, Mar. 2000. IEEE.

[9] J. Griffioen and R. Appleton. Reducing file system latency using a predictive approach. In *Proceedings of the Summer 1994 USENIX Technical Conference*, pages 197–207. USENIX, 1994.

[10] T. M. Kroeger and D. D. E. Long. Predicting file–system actions from prior events. In *Proceedings of the Winter 1996 USENIX Technical Conference*, pages 319–328, San Diego, CA, Jan. 1996.

[11] T. M. Kroeger and D. D. E. Long. The case for efficient file access pattern modeling. In *Proceedings of the 7th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VII)*, pages 14–19, Rio Rico, AZ, Mar. 1999. IEEE.

[12] T. M. Kroeger and D. D. E. Long. Design and implementation of a predictive file prefetching algorithm. In *Proceedings of the 2001 USENIX Annual Technical Conference*, pages 105–118, Boston, MA, Jan. 2001. USENIX.

[13] G. H. Kuenning, W. Ma, P. L. Reiher, and G. J. Popek. Simplifying automated hoarding methods. In *Proceedings of the 5th ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM'02)*, Atlanta, GA, Sept. 2002. ACM.

[14] G. H. Kuenning, G. J. Popek, and P. L. Reiher. An analysis of trace data for predictive file caching in mobile computing. In *Proceedings of the Summer 1994 USENIX Technical Conference*, pages 291–303. USENIX, 1994.

[15] H. Lei and D. Duchamp. An analytical approach to file prefetching. In *Proceedings of the 1997 USENIX Annual Technical Conference*, pages 275–288. USENIX, Jan. 1997.

[16] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, Feb. 1994.

[17] L. Mummert and M. Satyanarayanan. Long term distributed file reference tracing: Implementation and experience. *Software—Practice and Experience (SPE)*, 26(6):705–736, June 1996.

[18] J. K. Ousterhout. Why aren't operating systems getting faster as fast as hardware? In *Proceedings of the Summer 1990 USENIX Technical Conference*, pages 247–256. USENIX, 1990.

[19] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP '95)*. ACM, 1995.

[20] C. Ruemmler and J. Wilkes. UNIX disk access patterns. In *Proceedings of the Winter 1993 USENIX Technical Conference*, pages 405–420, San Diego, CA, Jan. 1993. USENIX.

[21] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

[22] J. S. Vitter and P. Krishnan. Optimal prefetching via data compression. *Journal of the Association for Computing Machinery*, 43(5):771–793, 1996.

[23] V. Vovk. Aggregating strategies. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, pages 371–383, Rochester, NY, 1990. Morgan Kaufmann.

[24] G. A. S. Whittle, J.-F. Pâris, A. Amer, D. D. E. Long, and R. Burns. Using multiple predictors to improve the accuracy of file access predictions. In *Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 230–240, Apr. 2003.

[25] W. A. Wulf and S. A. McKee. Hitting the memory wall: Implications of the obvious. *SIGARCH Computer Architecture News*, 23(1):20–24, 1995.

[26] T. Yeh, D. D. E. Long, and S. Brandt. Performing file prediction with a program–based successor model. In *Proceedings of the 9th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '01)*, pages 193–202, Cincinnati, OH, Aug. 2001. IEEE.