A Two-Expert Approach to File Access Prediction

Wenjing Chen Christoph F. Eick Jehan-François Pâris[1]
Department of Computer Science
University of Houston
Houston, TX 77204-3010
tigerchenwj@yahoo.com, {ceick, paris}@cs.uh.edu

Abstract

We present a new technique for reducing the number of false predictions made by file access predictors. Our Two-Expert Approach *combines the outcomes of two file access predictors, such as Stable Successor or Recent Popularity, and declines to make a prediction unless these two predictors agree. Experimental evidence shows that our two-expert approach produces significantly fewer false predictions than each of its two component predictors.*

1. Introduction

CPU speeds have been roughly doubling every other year over the last twenty years. Memory sizes and disk drive capacities have followed a similar evolution. Disk drive access times are the exception. They have only improved by a factor of 3 to 4 since the early eighties. As a result, disk access times have become a bottleneck in an increasing number of I/O intensive applications. The situation is not likely to improve soon because disk access times are limited by mechanical constraints that do not apply to solid state devices.

Two main techniques have been used to alleviate the problem, namely caching and prefetching. Caching keeps in memory the data that are the most likely to be used again while prefetching attempts to bring data in memory before they are needed. Both techniques have been widely used at the block level and start now to be applied at the file level. File-level prefetching is inherently more difficult to implement than file-level caching because prefetching files that are not needed can have a direct negative impact on system performance while keeping in a cache files that will not be reused only reduces the cache effectiveness. A key requirement for a successful implementation of file prefetching is thus a good file access predictor. This predictor should have reasonable space and time requirements, make as many correct predictions as possible and as few false predictions as feasible.

Early file access predictors [5-10, 12, 14-16] relied on sophisticated heuristics that required maintaining a large amount of information about past file references. More recent contributions [1, 2, 17] have shown that very simple predictors requiring much less context information could provide surprisingly accurate predictions.

A common limitation of these simple predictors is the relatively high number of false predictions they make. One possible way to reduce that number would be to develop more cautious predictors that would decline to make a prediction whenever they do not detect a strong access pattern and would thus be likely to make fewer false predictions. We propose a different solution: running two predictors in parallel, comparing the predictions they make and accepting these predictions if and only if they agree. As a result, this Two-Expert Approach will only return a wrong prediction when both its components agree on the same wrong prediction. Experimental evidence shows that our approach produces much less false predictions than each of its two component predictors.

The remainder of this paper is organized as follows. Section 2 reviews previous work. Section 3 introduces the performance criteria that we used in our study. Section 4 presents our Two-Expert Approach and compares its performance with that of other predictors. Finally, Section 5 states our conclusions.

2. Previous Work

Palmer and Zdonik [12] developed an associative memory that recognized access patterns within a context over time. Their predictive cache, named *Fido*,

[1] Supported in part by the National Science Foundation under grant CCR-9988390.

learns file access patterns within isolated access *contexts*. Griffioen and Appleton [6] presented in 1994 a file prefetching scheme relying on graph-based relationships. Their probability graphs only tracked the frequency of access within a particular "look-ahead" window size. Shriver *et al.* [14] proposed an analytical performance model to study the effect of prefetching for file system reads. Their model was based on a 4.4BSD-derived file system, and was validated against several simple workloads. Predictions of the model were found to be typically within 4 percent of measured values.

Tait *et al.* [15] investigated a client-side cache management technique for detecting file access patterns and for exploiting them to prefetch files from servers. They hypothesized that the work patterns of most individuals give rise to file access patterns that define working sets of files used for particular applications. Lei and Duchamp [10] later extended this approach and used a *match threshold* to quantify the degree of compatibility between stored *pattern trees* representing file working sets and the *working trees* being formed. These authors also introduced the *Last Successor* predictor, which takes the most recently observed successor of file *F* as the predicted successor of the next occurrence of *F*. More recent work by Kroeger and Long [9] compared the predictive performance of that predictor to that of Griffioen and Appleton's scheme and introduced more effective schemes based on context modeling and data compression.

Stable Successor (or Noah) [1] and *Recent Popularity* [2] extend the Last Successor predictor by attempting to filter out noise in the observed file reference stream. *Stable Successor* keeps track of the last observed successor of every file, but it does not update its past prediction of file the successor of a file *A* before having observed *s* successive instances of file *B* immediately following instances of file *A*. Hence, given the sequence

S: ABABABABACABACACABADADADA

Stable Successor with $s = 3$ will initially predict that file *B* is the successor of file *A* and will not update this prediction A until it encounters 3 consecutive instances of file D immediately following instances of file A. Increasing *s* from 3 to 4 would require 4, instead of 3, consecutive instances of file D immediately following instances of file A to update the predictor, thus increasing the stability of the algorithm and diminishing its responsiveness.

Figure 1 describes the algorithmic behavior of a Stable Successor predictor. We immediately observe that Stable Successor will either make predictions of guesses depending on its level of confidence:

Assumptions:
G is file being currently accessed
F its direct predecessor
StableSuccessor(*F*) is last prediction made for the successor of *F*
LastSuccessor(*F*) is last observed successor of *F*
Count(F) is a counter
s is the *stability* parameter of the algorithm
Algorithm:
if *LastSuccessor*(*F*) = *G* **then**
    Counter(*F*) ← *Counter*(*F*) + 1
else
    Counter(*F*) ← 0
end if
if *Counter*(*F*) ≥ *s* **then**
    LastSuccessor(*F*) ← G
    StableSuccessor(*F*) ←G
    Predict *StableSuccessor*(*F*)
else
    LastSuccessor(*F*) ← G
    Guess *StableSuccessor*(*F*)
end if

Figure 1. The Stable Successor predictor

(a) Stable Successor will *predict* that file *G* will the successor of file *F* whenever *G* was the observed successor of *F* for the last *s* instances of *F*; *G* will then become the new *stable successor* of *F*.

(b) Stable Successor will only make a *guess* for the successor of the *F* whenever the last *s* instances of *F* did not have the same successor. That guess will be the current *stable successor* of *F*.

Recent Popularity or *Best j-out-of-k* [2, 17] provides the stability benefits of Stable Successor while allowing for faster adaptation to workload changes. Recent Popularity keeps track of the last *k* most recently observed successors of a file. When attempting to make a prediction for file *F*, it searches for the most popular successor from the list. If the most popular successor *G* occurs at least *j* times then it is becomes the current *j-out-of-k successor* of *F*. When more than one file qualifies as the "most popular," recency is used as the tiebreaker. Whittle *et al.* [17] added a third parameter $l \leq j$ to let Recent Popularity predict the last successful *j-out-of-k* if it appears at least *l* times in the list of *k* last successors.

The *Composite File Predictor* [17] applies four independent heuristics to the same context information and select the one that is the most likely to deliver an accurate prediction. These four heuristics are (1) Stable Successor, (2) Predecessor Position, (3) Pre-

predecessor Position and (4) Recent Popularity. Given the sequence

S: ACFBDE<u>ABC</u>DFAB

Predecessor Position predicts that *C* will be the successor of *B* because the file reference sequence "*ABC*" occurred in the recent past. Pre-predecessor Position extends the same approach one step further. Given the sequence

S: ACFBDE<u>ABC</u>DFABC

Predecessor Position predicts that *D* will be the successor of *C* because the file reference sequence "*ABCD*" occurred in the recent past.

The two parameters of the Composite Predictor are a factor $\alpha \leq 1$ expressing the cost of a false prediction and the number *l* of previous successors it maintains foe each file (history length).

Finally, Brandt *et al.* [3] recently presented another composite predictor that combines multiple predictors or "experts" to reduce the number of false predictions. Their set of experts includes a *null* prediction expert that suppresses prefetching whenever the likelihood of an accurate prefetch is low.

3. Performance Evaluation Issues

A good file predictor should make as many correct predictions as possible and as few false predictions as feasible. This is to say that the performance of a file access predictor cannot be easily reduced into a single number. Several complementary metrics have been proposed, among which *accuracy*, *coverage*, and *success-per-reference rate.*

Consider a file predictor making N_{corr} correct predictions and N_{incorr} incorrect predictions when applied to N_{ref} file references. Observe that $N_{corr} + N_{incorr} \leq N_{ref}$ because the predictor can decline to make a prediction whenever it does not detect a file access pattern. We define the *accuracy a* of the predictor as its success-per-prediction ratio:

$$a = \frac{N_{corr}}{N_{corr} + N_{incorr}}$$

and its *coverage c* by its prediction-per-reference ratio:

$$c = \frac{N_{corr} + N_{incorr}}{N_{ref}} .$$

The *success-per-reference rate s* of the predictor is then given by:

$$s = \frac{N_{corr}}{N_{ref}} = ac .$$

Assumptions:
F is file being currently accessed
G_1 is the prediction of the successor of *F* made by the first component predictor
G_2 is the prediction of the successor of *F* made by the second component predictor
Algorithm:
if $G_1 = G_2$ **then**
    predict G_1
else
    issue no prediction
end if

Figure 2. The Two-Expert Approach

Because of the dependent nature of these three metrics, it is not possible to use anyone of them alone when assessing the performance of any given predictor. For example, a predictor that has 99 percent *accuracy* would be useless if it could only be used on 5 percent of the references. Conversely, a predictor that has a high *coverage* may also give make an unacceptably high number of incorrect predictions. Another disadvantage of these three metrics is that they discount the performance improvement achieved by increasing the success rate of a predictor from, say, 86 to 93%. It makes this improvement appear marginal, even though it represents a 50% reduction in the number of misses.

Whittle et al. [17] proposed a fourth metric that captures both aspects of the predictor performance, the *effective-miss-ratio* which is the ratio:

$$\frac{N_{ref} - N_{corr} + \alpha N_{incorr}}{N_{ref}}$$

A zero value for α corresponds to the ideal situation where false predictions would incur no penalty because all predicted file fetches could be instantly preempted when found to be incorrect. A unit value assumes that there all ongoing fetches must be completed, whether correctly predicted or not. The penalty for the incorrect prediction is then equivalent to one additional cache miss. Intermediate α values represent situations where preemption will reduce but not eliminate the penalty for a false prediction.

4. Our Two-Expert Approach

Our premise is that any predictor having a reasonable coverage will fail from time to time to predict the correct successor of the current file. Our objective is to reduce this number of false predictions. Rather than trying to increase the accuracy of an existing predictor

Table I. The twelve component predictors

Predictor	*Predictor Parameters*			*Coverage*	*Accuracy*	*Effective Miss Ratio*		
	s					**α=0**	**α=0.5**	**α=1**
Stable Successor	2			94.35%	73.51%	30.38%	42.75%	55.11%
	3			90.90%	74.42%	32.01%	43.47%	54.93%
	4			88.18%	74.95%	33.51%	44.36%	55.21%
Recent Popularity	***k***	***j***	***l***					
	6	5	3	71.28%	88.72%	36.41%	40.26%	44.11%
	7	5	3	75.28%	86.30%	34.64%	39.60%	44.55%
	9	8	5	65.76%	91.11%	39.78%	42.55%	45.31%
	10	8	5	68.28%	89.94%	38.27%	41.55%	44.83%
Composite Predictor	**α**	***History Length***						
	0	8		97.87%	77.86%	23.71%	34.50%	45.30%
	0.5	10		85.41%	85.96%	26.39%	32.29%	38.19%
	1	2		71.38%	89.72%	35.76%	39.32%	42.89%
	1	8		83.38%	87.15%	27.15%	32.42%	37.68%
	1	10		83.88%	87.24%	26.64%	31.89%	37.15%

by tightening its prediction criterion, we propose to run in parallel two different predictors, compare their outputs and accept these if and only if they agree. Figure 2 displays a more formal description of our algorithm. The outcome of this *Two-Expert Approach* (TEA) is a predictor that will only return a wrong prediction when both its components agree on the same wrong prediction.

Given the simplicity of the algorithm, its performance will be determined by the characteristics of its two component predictors. They should have both wide coverage and high accuracy. They should make a sufficient number of different predictions. Finally, they should have reasonable run-time overheads.

The two first criteria are obvious. The two component predictors should have wide coverages with a sufficient amount of overlap between themselves since the coverage of the TEA predictor will never be larger than that overlap. They should have high accuracies to reduce the likelihood that they would make the same wrong prediction. The two component predictors should also return a sufficient number of different predictions as there is no point in running two component predictors that always make the same predictions. Finally, the two component predictors should have reasonable run-time overheads. We have to consider here both their space and their time overheads. The space overheads of the predictors will depend on the amount of history kept for each file while their time overheads will represent the cost of maintaining and analyzing these data. We will thus restrict our search to simple predictors that maintain a limited amount of context information for each file. To reduce even more the space overhead, we should give preference to predictors that share the same context information.

4.1. Selecting the component heuristics

We restricted our search to three groups of predictors, namely Stable Successor [1], Recent Popularity [2] and the more recent Composite Predictor [17] because these predictors offered a wide coverage and a high accuracy, had a low overhead and shared the same context information.

To evaluate the performance of the candidate predictors and to measure how frequently they diverged, we simulated their execution over on two sets of file traces. The first set consisted of four file traces collected using Carnegie Mellon University's DFSTrace system [11]. The traces include *mozart*, a personal workstation, *ives*, a system with the largest number of users, *dvorak*, a system with the largest proportion of write activity, and *barber*, a server with the highest number of system calls per second. These traces provide information at the system-call level, and represent the original stream of access events not

Table II. Applying our approach to two Stable Successor predictors.

s	*s*	*Coverage*	*Accuracy*	*Effective Miss Ratio*		
				α=0	**α=0.5**	**α=1**
2	3	83.03%	80.70%	33.86%	40.32%	46.77%
2	4	80.79%	82.28%	35.54%	40.62%	45.70%
3	4	83.85%	78.96%	34.41%	42.18%	49.95%

Table III. Applying our approach to a Stable Successor and a Recent Popularity predictor.

Stable Successor	*Recent Popularity*			*Coverage*	*Accuracy*	*Effective Miss Ratio*		
s	*k*	*j*	*l*			**α=0**	**α=0.5**	**α=1**
2	6	5	3	69.59%	89.91%	37.14%	40.51%	43.88%
2	7	5	3	72.53%	88.21%	35.71%	39.83%	43.95%
2	9	8	5	64.74%	92.12%	40.10%	42.51%	44.93%
2	10	8	5	66.61%	91.26%	38.95%	41.72%	44.50%
3	6	5	3	70.36%	89.36%	36.81%	40.39%	43.97%
3	7	5	3	73.02%	87.70%	35.60%	39.91%	44.22%
3	9	8	5	65.26%	91.61%	39.92%	42.51%	45.10%
3	10	8	5	67.41%	90.63%	38.61%	41.62%	44.63%
4	6	5	3	70.24%	89.32%	36.93%	40.52%	44.10%
4	7	5	3	72.43%	87.85%	36.00%	40.21%	44.42%
4	9	8	5	65.39%	91.44%	39.91%	42.56%	45.21%
4	10	8	5	67.59%	90.40%	38.58%	41.67%	44.76%

filtered through a cache. They include between four and five million file accesses. Our second set of traces was collected in 1997 by Roselli [13] at the University of California, Berkeley over a period of approximately three months. To eliminate any interleaving issues, these traces were processed to extract the workloads of an instructional machine (*instruct*), a research machine (*research*) and a web server (*web*).

Table I summarizes the coverages, accuracies and effective miss ratios of the twelve predictors we selected. These values are averages over the seven traces as space considerations prevented us from displaying more detailed results.

4.2. Applying the TEA to two Stable Successor predictors

We investigated first applying our approach to pairs of Stable Successor predictors. The results are summarized in Table II. As before, the results are average results over the seven traces. Comparing the data from Tables I and II, we can see that the TEA produces predictors that deliver lower effective miss ratios whenever false predictions are penalized.

4.3. Applying the TEA to Stable Successor and Recent Popularity predictors

Table III summarizes our findings. Combining a Stable Successor and a Recent Popularity predictor can bring the accuracy of the predictions above 90 percent. The results for the effective miss ratio are less conclusive as they are not better than those achieved by Recent Popularity with $k = 10$, $j = 8$ and $l = 5$. The explanation to this apparent paradox lies in the relatively low coverage of the TEA predictors. Even though they are more accurate than the best Recent Popularity Predictors, they also make less correct predictions, which results in higher effective miss ratios. We should also note that the TEA applied to

Table IV. Applying our approach to a Stable Successor and a Composite predictor.

Composite		*Stable Successor*	*Coverage*	*Accuracy*	*Effective Miss Ratio*		
History Length	*α*	*s*			**α=0**	**α=0.5**	**α=1**
8	0.0	2	74.33%	88.89%	33.71%	37.72%	41.74%
10	0.5	2	71.30%	91.20%	34.78%	37.82%	40.86%
2	1.0	2	64.67%	93.72%	39.23%	41.18%	43.14%
8	1.0	2	70.56%	91.54%	35.23%	38.12%	41.01%
10	1.0	2	71.00%	91.43%	34.90%	37.86%	40.81%
8	0.0	3	70.99%	91.03%	35.18%	38.27%	41.35%
10	0.5	3	69.50%	92.16%	35.77%	38.40%	41.04%
2	1.0	3	63.41%	94.54%	39.91%	41.57%	43.24%
8	1.0	3	68.80%	92.50%	36.19%	38.68%	41.17%
10	1.0	3	69.29%	92.32%	35.86%	38.44%	41.01%
8	0.0	4	68.86%	91.90%	36.54%	39.24%	41.94%
10	0.5	4	67.81%	92.74%	36.95%	39.32%	41.70%
2	1.0	4	62.25%	94.85%	40.82%	42.36%	43.89%
8	1.0	4	67.18%	93.04%	36.84%	39.10%	41.35%
10	1.0	4	67.66%	92.85%	37.01%	39.35%	41.68%

Table V. Applying our Approach to Two Composite Predictors.

Composite		*Composite*		*Coverage*	*Accuracy*	*Effective Miss Ratio*		
History Length	*α*	*History Length*	*α*			**α=0**	**α=0.5**	**α=1**
8	0.0	10	0.5	84.68%	86.13%	26.86%	32.64%	38.41%
8	0.0	2	1.0	70.25%	91.01%	35.88%	38.94%	42.00%
8	0.0	8	1.0	83.38%	87.15%	25.68%	30.95%	36.22%
8	0.0	10	1.0	83.17%	87.43%	27.10%	32.24%	37.38%
10	0.5	2	1.0	69.91%	91.35%	35.94%	38.87%	41.80%
10	0.5	8	1.0	82.89%	87.57%	27.23%	32.29%	37.35%
10	0.5	10	1.0	83.88%	87.24%	26.64%	31.89%	37.15%
2	1.0	8	1.0	70.23%	91.02%	35.88%	38.94%	42.00%
2	1.0	10	1.0	69.91%	91.35%	35.94%	38.87%	41.80%
8	1.0	10	1.0	82.89%	87.57%	25.99%	31.05%	36.11%

Stable Successor and Recent Popularity predictors performs better than the TEA applied to Stable Successors alone.

4.4. Applying the TEA to Stable Successor and Composite predictors

Our highest accuracies were achieved by applying our approach to a Stable Successor and a Composite predictor. As we can see in Table IV, a TEA combining these two predictors can reach up to 94.84 percent accuracies over a very wide range of traces reflecting very different user behaviors. These excellent results do not translate into lower effective miss rates because of the accompanying decrease in the coverage of the predictor.

4.5 Applying the TEA to two Composite predictors

Our lowest effective miss ratios were obtained by applying our approach to pairs of Composite Predictors. Comparing the results displayed in Tables I and V, we can see that a TEA combining two Composite Predictors can achieve lower effective miss rates than the best Composite Predictor. While the margin is very narrow, the improvement is still noteworthy because we are comparing a generic TEA predictor with two Composite Predictors reaching they best performance for the α value for which they were tuned.

We also applied our approach to pairs of Stable Successor and Recent Popularity predictors as well as for pairs of Recent Popularity and Composite predictors. We did not include them, as they were very similar to the results we reported.

5. Conclusion

We have presented a Two-Expert Approach (TEA) aiming at improving the accuracy of file access predictors. Rather than relying on a single file prediction heuristic, the TEA predictor combines the outcomes of two file access predictors and declines to make a prediction unless these two predictors agree. As a result, our predictor will only return a wrong prediction when both its components agree on the same wrong prediction. Prediction overhead can be kept low by selecting component predictors that maintain a limited amount of context information.

We found out that the highest accuracies were obtained by combining the outcomes of a Stable Successor and a Composite Predictor. The best effective miss ratios, that is, miss ratios taking into account the negative impact of false predictions, were achieved by combining the outcomes of two Composite Predictors.

In addition, our study confirms the excellent performance of the Composite Predictor and indicates a need for more sophisticated methods for combining the outcomes of several predictors.

References

[1] A. Amer and D. D. E. Long, Noah: Low-cost file access prediction through pairs, in *Proc. 20th International Performance, Computing, and Communications Conference*, pp. 27–33, April 2001.

[2] A. Amer, D. D. E. Long, J.-F. Pâris, and R. C. Burns, File access prediction with adjustable accuracy, in *Proc. 21st International Performance of Computers and Communication Conference*, pp. 131-140, April 2002.

[3] K. Brandt, D. D. E. Long and A. Amer, Predicting When Not To Predict, in *Proc. 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '04)*, Oct. 2004,.

[4] I. C. K. Chen, J. T. Coffey, and T. N. Mudge, Analysis of branch prediction via data compression, in *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 128–137, Oct. 1996.

[5] K. M. Curewitz, P. Krishnan, and J. S. Vitter, Practical prefetching via data compression, in *Proc. 1993 ACM SIGMOD Conference on Management of Data*, pp. 257–266, May 1993.

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

[7] P. Krishnan, Online prediction algorithms for databases and operating systems, PhD Thesis, Dept. of Computer Science, Brown University, 1995.

[8] T. M. Kroeger and D. D. E. Long, The case for efficient file access pattern modeling, in *Proc. 1996 USENIX Technical Conference*, pp. 14–19, Jan. 1996.

[9] T. M. Kroeger and D. D. E. Long, Design and implementation of a predictive file prefetching algorithm, in *Proc. 2001 USENIX Annual Technical Conference*, pp. 105–118, June 2001.

[10] H. Lei and D. Duchamp, An analytical approach to file prefetching, in *Proc. 1997 USENIX Annual Technical Conference*, Jan. 1997.

[11] L. Mummert and M. Satyanarayanan, Long term distributed file reference tracing: implementation and experience, Technical Report, School of Computer Science, Carnegie Mellon University, 1994.

[12] M. L. Palmer and S. B. Zdonik, FIDO: a cache that learns to fetch, in *Proc. 17th International Conference on Very Large Data Bases*, pp. 255–264, Sept. 1991.

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

[14] E. Shriver, C. Small, and K. A. Smith, Why does file system prefetching work? in *Proc. 1999 USENIX Technical Conference*, pp. 71–83, June 1999.

[15] C. Tait and D. Duchamp, Detection and exploitation of file working sets, in *Proc. 11th International Conference on Distributed Computing Systems*, pp. 2–9, May 1991.

[16] J. S. Vitter and P. Krishnan, Optimal prefetching via data compression, in *Proc. 32nd Annual IEEE Symposium on Foundations of Computer Science*, pp. 121–130, Oct. 1991.

[17] 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 *Proc. 20th IEEE Symposium on Mass Storage Systems* (MSS 2003), pp. 230–240, April 2003.