Title

A stochastic approach to file access prediction

Authors

Pâris, Jehan-François
Amer, Ahmed
Long, Darrell DE

Publication Date

2003

DOI

10.1145/1162618.1162623

A Stochastic Approach to File Access Prediction

Jehan-François Pâris, Ahmed Amer, Darrell D. E. Long

Abstract—Most existing studies of file access prediction are experimental in nature and rely on trace driven simulation to predict the performance of the schemes being investigated. We present a first order Markov analysis of file access prediction, discuss its limitations and show how it can be used to estimate the performance of file access predictors, such as *First Successor*, *Last Successor*, *Stable Successor* and *Best-k-out-of-n*. We compare these analytical results with experimental measurements performed on several file traces and find out that specific workloads, and indeed individual files, can exhibit very different levels of non-stationarity. Overall, at least 60 percent of access requests appear to remain stable over at least a month.

Index Terms—File access prediction, storage hierarchies, storage management.

1 Introduction

ONE of the most difficult problems facing operating systems designers is finding the best way to manage memory hierarchies consisting of devices with widely different access times. The problem is not new and is worsening as gains in main memory access times have dramatically outpaced gains in disk access times.

Two main techniques can be used to mitigate this 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. Prefetching is inherently more difficult to implement than caching because prefetched data that are not needed can have a direct negative impact on system performance while keeping in a cache data that will not be reused only reduces the cache effectiveness. As a result, most systems err on the side of caution and do not exploit the full potential of the technique.

A key requirement for a successful implementation of file prefetching is a good *file access predictor*. This predictor should have reasonable space and time requirements, make as many successful predictions as possible and as few bad predictions as feasible.

Most existing studies of file access prediction have been experimental in nature, relying on trace driven simulation to predict the performance of individual predictors. Even today, no comprehensive probabilistic analysis of file predictors can be found in the literature.

This omission is especially regrettable given the very good performance of some very simple predictors such as *First Successor*, *Last Successor*, *Stable Successor* and *Best-k-out-of-n*. These predictors base all their prediction of the next file to be accessed on the identity of its immediate predecessor. In that sense, they implicitly assume that file access patterns can be modeled by a first-order Markov model.

This observation motivated us to present a first order Markov analysis of performance of these file access predictors and discusses its limitations, among which its incapacity to represent non-stationary behaviors.

The remainder of this paper is organized as follows. Section 2 reviews previous work on file access prediction. Section 3 introduces our model and Section 4 discusses its limitations. Finally, Section 5 states our conclusions.

2 Previous Work

Palmer *et al*. [9] used an associative memory to recognize access patterns within a context over time. Their predictive cache, named *Fido*, learns file access patterns within isolated access *contexts*. Griffioen and Appleton [4] 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*. [11] proposed an analytical performance model to study the effect of prefetching for file system reads. The 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*. [12] investigated a client-side cache management technique used 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 [7] 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 *A* as the predicted successor of the next occurrence of *A*.

- *Jehan-François Pâris is with the Computer Science Department, University of Houston, Houston, TX 770204-3010. E-mail: paris@cs.uh.edu.*
- *Ahmed Amer. is with the Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: amer@cs.pitt.edu.*
- *Darrell D. E. Long is with the Computer Science Department, University of California, Santa Cruz, CA 95064. E-mail:darrell@cs.ucsc.edu.*

More recent work by Kroeger and Long [5, 6] 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. Finally, Yeh *et al*. [15] investigated a simple but effective successor model that identifies the relationships between files through identification of the programs accessing them.

Two predictors *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* ignores observations that vary too rapidly, effectively acting as a low-pass filter.

Stable Successor keeps track of the last observed successor of every file, but it does not update its past prediction of the successor of file *X* before having observed $s + 1$ successive instances of file *Y* immediately following instances of file *X*. Hence, given the sequence:

S: ABABACABACACABADADADA

Stable Successor with $s = 2$ will not update its predictor the successor of A until it encounters 3 consecutive instances of file *D* immediately following instances of file *A*. Increasing *s* from 2 to 3, 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* or *guesses* depending on its level of confidence:

1) Stable Successor will *predict* that file *G* will be the successor of file *F* whenever *G* was the observed successor of *F* for the last $s + 1$ instances of *F*; *G* will then become the new *stable successor* of *F*.

2) Stable Successor will only make a *guess* for the successor of file *F* whenever the last $s + 1$ instances of *F* did not have the same successor. That guess will be the current *stable successor* of *F*.

Recent Popularity or *Best k-out-of-m* provides the stability benefits of Stable Successor while allowing for faster adaptation to workload changes. Best *k*-out-of-*m* keeps track of the last *m* most recently observed successors of a file. When attempting to make a prediction for a given file, *recent popularity* searches for the most popular successor from the list. If the most popular successor occurs at least *k* times then it is submitted as a prediction. When more than one file qualifies as "most popular," recency is used as the tiebreaker.

Table I summarizes the performances of these three predictors and compares them to those of a very simple predictor that always selects the first observed successor to a file (*First Successor*). The seven traces used in these experiments belonged to two sets of traces. A first set consisted of four file traces collected using Carnegie Mellon University's *DFSTrace* system [8]. The traces include *mozart*, a personal workstation,

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

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. They include between four and five million file accesses. A second set of traces was collected in 1997 by Roselli [10] 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*).

We can make a few general observations about these data. First, the successor to any given file can be predicted with 70 to 80 percent accuracy without any reference to a larger context. Second, predictors that base their predictions of the next successor to a given file on the identities of the most recently observed successors to the file fare much better than predictors using older observations.

3 Our Model

A first-order Markov model for file access assumes that the next file to be referenced only depends on its immediate predecessor. Thus file *j* will be accessed after file *i* with conditional probability p_{ij} and the steady state probability p_j of accessing file *j* is given by the system of *n* linear equations:

$$p_j = \sum_{i=1}^{n} p_i p_{ij}$$

where *n* is the total number of files to be considered. We both have

TABLE I. HIT RATIOS OF SOME FILE ACCESS PREDICTORS (FROM [13, 14])

Algorithm	Barber (%)	Dvorak (%)	Ives (%)	Mozart (%)	Instruct (%)	Research (%)	Web (%)
First Successor	55.1	27.65	32.59	55.2	10.47	32.32	28.3
Last Successor	77.76	67.61	65.2	74.45	72.43	56.28	42.88
Stable Successor	81.01	71.71	70.47	77.97	76.35	58.3	51.53
Best k-out-of-m	81.36	72.59	70.85	78.46	76.82	61.31	51.39

$$\sum_{i=1}^{n} p_i = 1$$

and

$$\sum_{j=1}^{n} p_{ij} = 1 \text{ for } 1 \le i \le n .$$

Let us consider first the performance of the *First Successor* predictor. The conditional probability that file j is the successor of file i is given by p_{ij}. Thus the chance of having file j as first successor of file i is also given by p_{ij}.

The probability of making a correct prediction will depend on (a) the probability p_{ij} of predicting a specific file j as successor to file i and (b) the probability p_{ij} that this prediction will materialize. Since both events are independent, we can multiply the two probabilities and the overall probability of making a correct prediction for the successor of file i is then given by

$$\sum_{j=1}^{n} p_{ij}^2 .$$

Summing over all possible predecessors, we find that the overall success ratio of the predictor is given by

$$\sum_{i=1}^{n}\sum_{j=1}^{n} p_i p_{ij}^2 .$$

Since the predictor always makes a prediction, its probability of making a bad prediction is given by

$$1 - \sum_{i=1}^{n}\sum_{j=1}^{n} p_i p_{ij}^2 .$$

Under our assumptions, the performance of the *Last Successor* predictor should be not different from the performance of the *First Successor* predictor. This should not surprise us, as we base in both cases our prediction on one single previous observation. Hence the observed difference between the performance of the first successor and last successor predictors is a good indication that the process is not stationary and the p_{ij}'s vary over time.

Consider now a *Stable Successor* predictor that predicts file j to be the successor of i if the last $k = s + 1$ instances of file i were all followed by an instance of file j. The probability of predicting that file j is the successor of file i is then be given by $p_{ij}^{\,k}$ and the overall probability of making a correct prediction for the successor of file i is given by

$$\sum_{j=1}^{n} p_{ij}^{k+1}$$

Summing over all possible predecessors, we find that the overall success ratio of the predictor is given by

$$\sum_{i=1}^{n}\sum_{j=1}^{n} p_i p_{ij}^{k+1} .$$

Note that this predictor will decline to make a prediction whenever one or more of the last k instances of file i was not followed by an instance of file j. The probability of not making a prediction will thus be given by

$$\sum_{i=1}^{n}\sum_{j=1}^{n} p_i (1 - p_{ij}^{k}) .$$

The accuracy of the predictor will then be given by

$$\frac{\sum_{i=1}^{n}\sum_{j=1}^{n} p_i p_{ij}^{k+1}}{\sum_{i=1}^{n}\sum_{j=1}^{n} p_i (1 - p_{ij}^{k})} .$$

As two of the authors found out, one can increase the number of successful predictions of the protocol by "guessing" file j when the last sequence of k consecutive identical successors of file i contained file j [1]. Unfortunately, this extension makes the stable successor difficult to analyze. For the sake of simplicity, we will consider instead a somewhat more restrictive predictor that will only issue a guess when that last sequence of k consecutive identical successors occurred less than k reference pairs before.

For any $m < k$, the probability that the last sequence of k consecutive instances of file j succeeding file i occurred exactly m reference pairs before the current reference is given by

$$p_{ij}^{k}(1 - p_{ij}) .$$

Hence the probability that the last sequence of k consecutive identical successors occurred less than k reference pairs before is given by

$$p_{ij}^{k} + \sum_{m=1}^{k-1} p_{ij}^{k}(1 - p_{ij}) = k p_{ij}^{k} - (k-1) p_{ij}^{k+1} .$$

The probability of making a correct prediction for the successor of file i is thus given by

$$k \sum_{j=1}^{n} p_{ij}^{k+1} - (k-1) p_{ij}^{k+2} .$$

that is, the file with the highest conditional probability to succeed the current file. The probability of making a correct prediction for the successor of file i will then be given by the same $p_{i,j\max}$.

Summing over all possible predecessors, we find that the overall success ratio of the predictor is given by

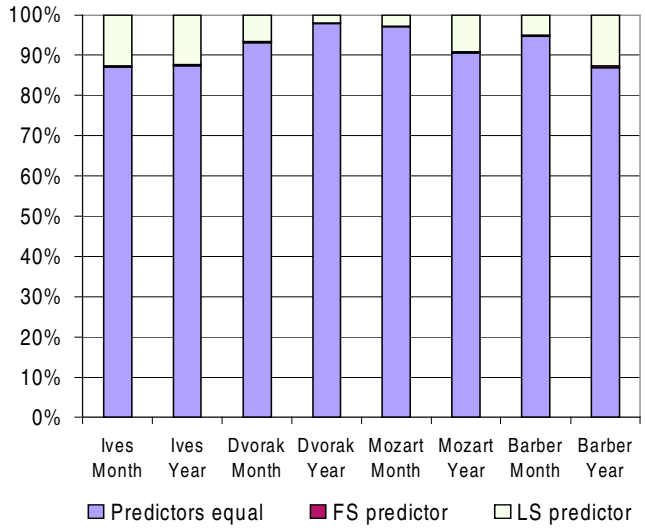

Figure 2 Compared performances of the First Successor (FS) and Last Successor (LS) on a per file basis.

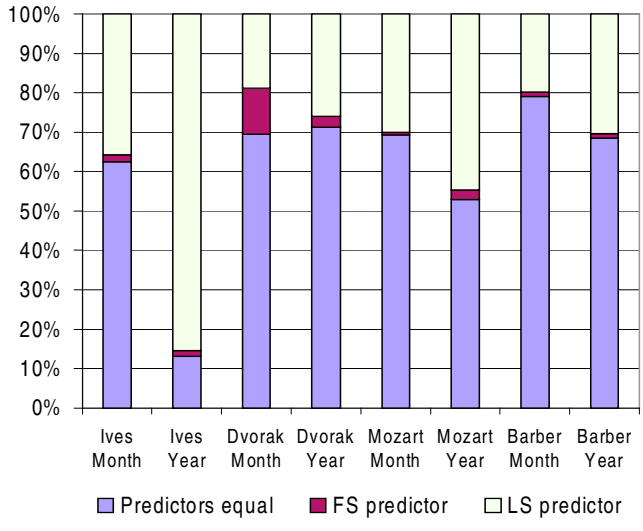

Figure 3 Compared performances of the First Successor (FS) and Last Successor (LS) on a per access basis.

$$k\sum_{i=1}^{n}\sum_{j=1}^{n} p_i p_{ij}^{k+1} - (k-1)\, p_{ij}^{k+2} .$$

Consider now a k-out-of-m predictor predicting that file j is the successor of file i whenever file j was the successor of file i in at least k of the last m instances of file i. The probability of predicting that file j is the successor of file i will be given by

$$\sum_{l=k}^{m}\binom{m}{k} p_{ij}^{l}(1-p_{ij})^{\text{k-1}} .$$

The probability of making a correct prediction for the successor of file i is then given by

$$\sum_{j-1}^{n}\sum_{l=k}^{m}\binom{m}{k} p_{ij}^{l+1}(1-p_{ij})^{\text{k-1}} .$$

Summing over all possible predecessors, we find that the overall success ratio of the predictor is given by

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=k}^{m}\binom{m}{k} p_i p_{ij}^{l+1}(1-p_{ij})^{\text{k-1}} .$$

As before, we could increase the number of correct predictions by continuing to predict file j as long as no other file is the successor of file i in at least k of the last m instances of file i.

Finally, our model suggests a fourth predictor, which we will call *Most Likely Successor.* It predicts that the successor of file i will be the file $j$max such that

$$p_{i,j\max} = \max\{\ p_{ij} \mid 1 \le j \le n\} ,$$

Summing over all possible predecessors, we find that the overall success ratio of the predictor is given by

$$\sum_{i=1}^{n} p_i p_{i,j\max} .$$

A practical implementation of this predictor would count the number of times any given file j has succeeded any given file i and always predict the most frequent successor of file i as its most likely successor.

4. DISCUSSION

Our Markov model makes two important assumptions. First, it assumes that the next file to be referenced only depends on its immediate predecessor. Second, it assumes that the probability p_{ij} of having file j succeeding file i does not vary over time. Let us briefly discuss these two assumptions.

Looking back at Table I, we see that all four predictors mentioned in the table base their predictions on the past successor history of the last file to be accessed. None of them makes any attempt to keep track of the predecessors of that file. Hence, the excellent performance of these four protocols is a strong indication that we only need to know the past successors of the last file to be accessed to be able to predict its successor.

Our hypothesis that probability p_{ij} of having file j succeeding file i does not vary over time is less easy to justify. If it were true, the First Successor and the Last Successor predictors should have identical success ratios. As shown on Table I, this is clearly contradicted by our experimental data, which indicate that Last Successor performs much better than First Successor for all seven traces. We should also observe that the margin of superiority of Last Successor over First Successor widely varies among the traces. While the hit ratio of the Last Successor predictor is 30 percent better than that of First Successor for the *mozart* trace, the same hit ratio is 592 percent better than that of First Successor for the *instruct* trace.
To gain a deeper understanding of this behavior, we need to examine the behavior of the two predictors on a per-file basis. Figure 2 shows how the different predictors performed on a per-file basis for the four CMU traces at two different durations, namely approximately a month and a year. (We could not include the three Berkeley traces in our study, as they were collected over a much shorter time span.)

Each bar shows the fraction of files for which both predictors performed equally well ("Predictors equal") as well as the fractions for which either Last Successor (LS) or First Successor (FS) was better. We can immediately see that both predictors performed identically for over eighty percent of the files, indicating that a large percentage of files have stable predictors. There are very few files for which the adaptive Last Successor predictor actually performed worse than the static First Successor predictor, with the remainder of all files showing a clear advantage for using the Last Successor.

A somewhat different picture emerges when we weight these results by the relative access frequencies of the files. As Figure 3 shows, there are then much fewer instances in which the two predictors are equivalent. In other words, many of the files for which they differ are accessed frequently. In particular the yearlong trace for *ives* shows that most accesses occur for files whose successors will vary over time. The dramatic shift between *ives-month* and *ives-year* supports the view that the non-stationary behavior for this particular workload is caused by a shift of successor behavior over time. Conversely, our measurements for the *dvorak* trace indicate that 70 percent of access requests remain stable over extended periods of time.

5. CONCLUSIONS

We have presented here a first order Markov analysis of file access prediction and show how it can be used to estimate the performance of file access predictors, such as *First Successor*, *Last Successor*, *Stable Successor* and *Best-k-out-of-m*.
We compared our analytical results with experimental measurements performed on several file traces and found out that individual workloads, and indeed individual files, could exhibit very different levels of non-stationarity. While we found yearlong stable access patterns for over 80 percent of all files accessed in the CMU traces, we also observed that these files were often less frequently accessed than the files with non-stationary access patterns. Even then, at least 60 percent of access requests appear to remain stable over at least a month.

The existence of stable access patterns among a large fraction of files in most workloads is a strong argument in favor of the development of more techniques aiming at clustering together files that are likely to be accessed together [3] as these clusters will often be able to remain unchanged over several weeks, if not several months.

ACKNOWLEDGMENTS

J.-F. Pâris was supported in part by the National Science Foundation under grant CCR-9988390. Ahmed Amer was supported in part by the National Science Foundation under grant ITR-0325353. Darrell D. E. Long was supported in part by the National Science Foundation under grant CCR-0204358.

REFERENCES

[1] A. Amer and D. D. E. Long, Noah: Low-Cost File Access Prediction Through Pairs, *Proc. 20th Int. Performance, Computing, and Communications Conf.*, 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, *Proc. 21st Int. Performance of Computers and Communication Conf.*, pp. 131-140, Apr. 2002.

[3] A. Amer, D. D. E. Long, and R. C. Burns. Group-based management of distributed file caches. *Proc. 22nd Int. Conf. on Distributed Computing Systems*, pp. 525–534, July 2002.

[4] J. Griffioen and R. Appleton, Reducing file System Latency Using a Predictive Approach, *Proc. 1994 Summer USENIX Conf.*, pp. 197–207, 1994.

[5] T. M. Kroeger and D. D. E. Long, The Case for Efficient File Access Pattern Modeling, *Proc. 1996 USENIX Technical Conf.*, pp. 14–19, Jan. 1996.

[6] T. M. Kroeger and D. D. E. Long, Design and Implementation of a Predictive File Prefetching Algorithm, *Proc. 2001 USENIX Annual Technical Conf.*, pp. 105–118, June 2001

[7] H. Lei and D. Duchamp, An Analytical Approach to File Prefetching, *Proc. 1997 USENIX Annual Technical Conf.*, Jan. 1997.

[8] L. Mummert and M. Satyanarayanan, Long Term Distributed File Reference Tracing: Implementation and Experience, Technical Report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1994.

[9] M. L. Palmer and S. B. Zdonik, FIDO: A Cache that Learns to Fetch, *Proc. Conf. on Very Large Data Bases (VLDB)*, Barcelona, pp. 255–264, September 1991.

[10] D. Roselli, Characteristics of File System Workloads, Technical. Report, University of California, Berkeley, 1998.

[11] E. Shriver, C. Small, and K. A. Smith, Why Does File System Prefetching Work? *Proc. 1999 USENIX Technical Conf.*, June 1999.

[12] C. Tait and D. Duchamp, Detection and Exploitation of File Working Sets, *Proc. 11th Int. Conf. on Distributed Computing Systems*, pp. 2–9, May 1991.

[13] G. A. S. Whittle, A Hybrid Scheme for File System Reference Prediction, MS Thesis, Department of Computer Science, University of Houston, Texas, May 2002.

[14] 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, *Proc. 20th IEEE Symp. on Mass Storage Systems & Technologies*, pp. 230–240, Apr. 2003.

[15] T. Yeh, D. D. E. Long, and S. Brandt, Performing File Prediction With a Program-Based Successor Model, *Proc. 9th Int. Symp. on Modeling, Analysis, and Simulation on Computer and Telecommunication Systems*, pp. 193–202, Aug. 2001.