Noah: Low-cost file access prediction through pairs

Ahmed Amer and Darrell D. E. Long[†]

Computer Science Department
Jack Baskin School of Engineering
University of California
Santa Cruz, CA 95064

Abstract

Prediction is a powerful tool for performance and usability. It can reduce access latency for I/O systems, and can improve usability for mobile computing systems by automating the file hoarding process. We present recent research that has resulted in a file successor predictor that matches the performance of state-of-the-art context-modeling predictors, while requiring a small fraction of their space requirements. Noah is an on-line algorithm for predicting successor file access events, effectively identifying strong pairings (successor relationships) among files. Noah can accurately predict approximately 80% of all file access events, while tracking only two candidate successors of which only one requires regular dynamic updates.

1. Introduction

We describe recent work that has resulted in an effective pair-wise access predictor – Noah – that demonstrates accuracy results comparable to, and in some cases better than, state-of-the-art dedicated predictive caching algorithms that require significantly more state. Among Noah's most notable features are: its usefulness for a variety of systems and data management problems, and its ability to perform with exceptionally low system overheads.

Latency is an ever-increasing component of data access costs, which in turn are usually the bottleneck for modern high performance systems. The ability to predict future data accesses is an important approach to addressing this growing problem. For this reason, accurate access predictors are very desirable for data storage systems. Another form of increased latency can be seen in mobile computing environments, where a disconnection is a potentially unbounded data access latency. If the requested data is not available locally, an unavoidable wait for the next available network connection (in the case of wireless networks this could be a weak and unreliable link), or the physical shipping of media, is incurred. Even if high-speed network connectivity is continuously available, traveling to a geographically distant location could impose significant latency. Even with the fastest networks, and best compression algorithms, a significant latency can be encountered by mobile users.

The ability to hoard data on a mobile computer's local storage goes a long way towards freeing the user from dependence on the (possibly non-existent) network connection. An important step in developing fully automated file hoarding algorithms is the ability to automatically identify strong relationships between files. The accurate prediction of the successor relationship between files is one such mechanism for identifying strong inter-file relationships. Prior works on mobile file hoarding such as Coda [4] and Seer [7] have made significant strides in automating the process of file hoarding, yet sufficient automation remains an elusive goal.

With systems handling much greater numbers of files and a growing volume of data, it is becoming more and more important to automate as many system tasks as possible. An algorithm that requires no manual parameter adjustments for "performance tweaking" is highly desirable. Noah addresses these issues by providing an algorithm that can accurately predict file access successors, using very limited state-space for each file, and requiring no manually adjusted operating parameters.

2. Successor Predictors

Our goal is to develop an accurate successor predictor that maintains strictly limited per-file state information. In Noah this involves tracking the last successor, and remembering a current prediction. No other information is used to make the prediction. Before discussing Noah, it is important to introduce some simple single-successor models, which maintain only one successor. We start by introducing

[†]Supported in part by the National Science Foundation award CCR-9972212, and the Usenix Association.

the distinction between dynamic and static predictors.

2.1. Dynamic and Static Predictors

In prior work we have observed that files can be divided into two categories, those that have a strong successor relationship to a fixed successor file, and those whose successor varies considerably for each file access. This fact was captured using two simple successor predictors, each of which only maintained one successor per file. The dynamic last-successor predictor, and the static first-successor model. The dynamic/last-successor model [6, 8] is updated with every access to a file to represent the dynamically observed last successor.

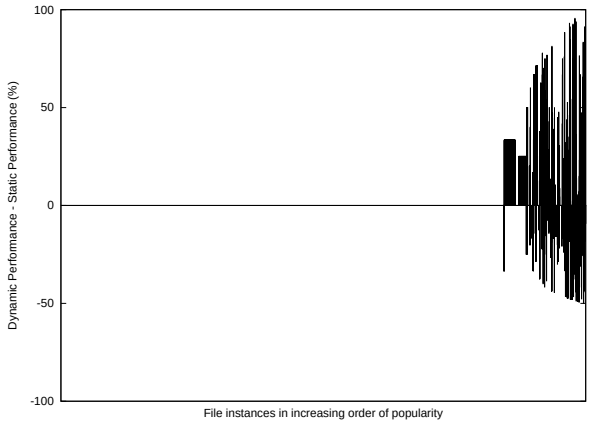
A static predictor, as its name implies, is not updated with observed file access events. The most important of these is the first-successor predictor, which maintains the first observed successor for a file as its unchanging successor prediction. In the same manner a second-successor predictor maintains the second unique file observed to succeed the current file as its unchanging predictor.

It might seem useful to track multiple static successors and somehow aggregate their performance to compare to the dynamic last-successor model, but through a tool we call the rank-difference plot it can be shown that second, third or fourth successors never outperform the first successor for any files accessed over the observation period.

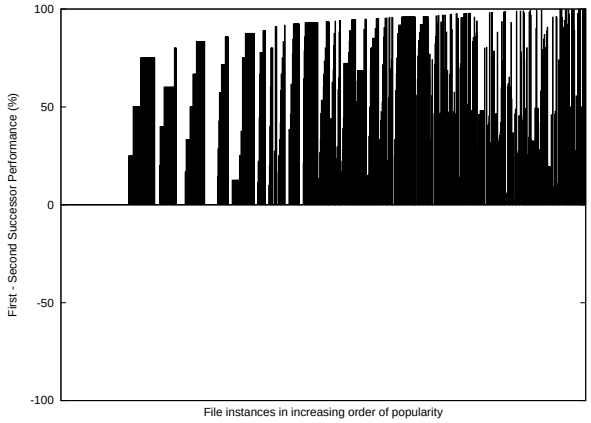
Rank-Difference Plots. A rank-difference plot represents a performance difference between two successor-predictors, plotted against instances of files ordered according to access frequency (increasing in the direction of the increasing x-axis). The y-axis on these graphs represents the difference between the accuracy (%) of two predictors, for the specific file at that point on the x-axis.

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 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.

Figure 1 shows the rank difference plots for a single system over a duration of approximately one month. Figure 1(a) compares the accuracy of the last-successor model to the static first-successor model on a per-file basis. Differences for the lower-frequency files are often non-existent. This is an artifact of the access-frequency distribution. Any improvement in predictions can only happen if a file is accessed more than once. If a file is accessed very few times,



(a) *barber* Last-Successor *vs.* First-Successor Accuracy



(b) *barber* First-Successor *vs.* Second-Successor Accuracy

Figure 1. Rank-difference plots for the single-successor models.

then any improvement in performance is subsequently limited.

Common intuition regarding file access behavior might suggest that, over extended periods of time, a static prediction can only compete on infrequently accessed files. Subsequently this assumption would conclude that a last-successor model would naturally perform better for all frequently accessed files. This assumption is invalid for every trace we have studied. In fact, there is a great variation between the accuracy of a single "first-guess" and a continuously updated guess. This suggests the existence of enough noise, in an access stream, to confuse a last-successor for a substantial number of file accesses that are best predicted as the first observed successor.

Figure 1(b) compares the accuracy of the first-successor

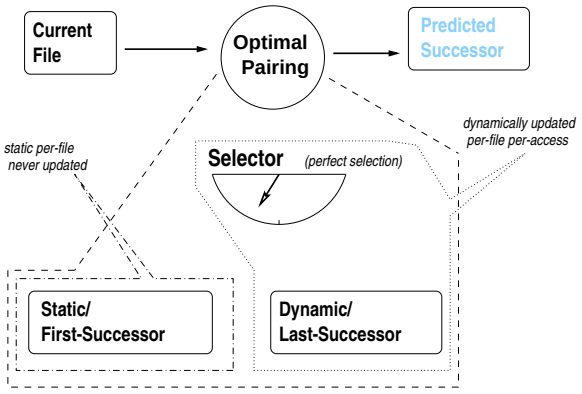


Figure 2. Conceptual View of the Optimal Pairing Algorithm

model to the second-successor model. The second-successor model associates each file with the second file observed as its successor. In other words, for the sequence **A B A C**, the second successor for **A** is **C**. The figure clearly demonstrates that the second successor is never found to be a better choice than the first successor, for any file over the trace period, Figure 1(b) is limited to only those files that have been accessed at least twice (otherwise they would not have a second successor), and that is the reason for the wider range of χ values (file instances) that are observed to have performance differences. Ideally, an optimal predictor that maintained a first and last successor prediction could produce a rank-difference plot like Figure 1(a) but with only positive performance values. We use such an algorithm for comparison against Noah. This "Optimal Pairing" algorithm is the subject of the following section.

2.2. Optimal Pairing

A conceptual view of an adaptive successor predictor that optimally combines both the last and first-successor models is given in Figure 2. In principle, any predictor that could maintain these two predictions per-file would have an upper bound on its accuracy dependent on the ability of the selector to choose between the two models. A perfect selector which always made the better choice would be equivalent to an on-line algorithm that could instantly adapt to the currently correct configuration.

It is reasonable to consider the application of machine learning algorithms such as the weighted majority algorithm [9] to select between these two models. In practice it was found that the simple filtered form of Noah, described in the following section, could almost match the performance of this optimal model. With very lengthy trace periods (over a year), it is sometimes possible for Noah to out-perform this two-expert model, thanks to its more adaptive nature. Implementing the optimal pairing algorithm is a simple matter of considering its prediction to be accurate if either of its experts is accurate (this models perfect selection/adaptation). This algorithm represents a higher goal than simply matching the correct predictor to the appropriate files, it is an optimal adaptive choice, which can be changed dynamically as new file access events are observed.

2.3. Noah

Figure 3(a) shows a simple conceptual view of Noah's operation, and Figure 3(b) illustrates Noah's update mechanism, indicating static elements, and those that require per-file-access updates. For each unique file Noah tracks the following information:

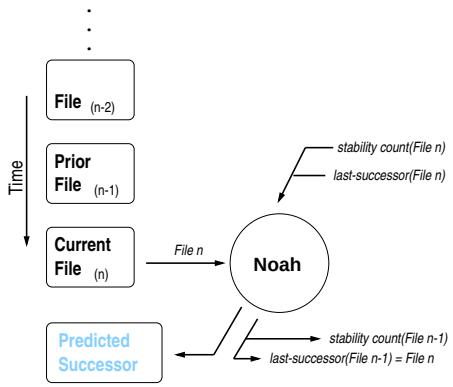
- **Current Prediction** – The current prediction. This is simply the first item observed to follow the current file. It is updated to match the candidate prediction (last-successor) if the candidate satisfies a stability condition.
- **Dynamic/Last-Successor** – The candidate prediction. This candidate is equivalent to the last successor model [6, 8], and is continuously updated with every access to a file to represent the dynamically observed last successor.

The main innovation of Noah is the extension of the basic last-successor predictor to filter out noise in the observed access stream. This is based on the intuition that noise is detrimental to the quality of any deductions made from dynamic observations. This filtered model effectively filters out observations that vary too rapidly, effectively acting as a low-pass filter for observations.

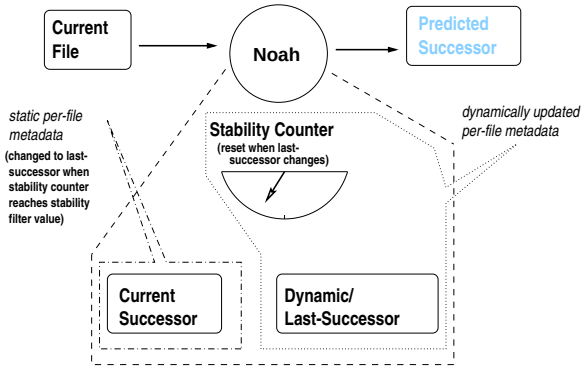
As with the last-successor model, this predictor maintains a record of the last file to be a successor to the current file. In addition to this, Noah maintains a current successor prediction, which is updated to become the last-successor, only if the last successor is observed to have remained consistently valid for a pre-determined stability count. While the last-successor updates its prediction with every access to a file, and the first successor never updates its initial prediction, Noah will update a file's successor if a new successor is observed consistently for a number of accesses.

The number of accesses in which a new successor must be observed to be correct, before the current successor is updated, is called the "stability" period. In this manner, files with successors that change rapidly, *i.e.* have no stable observed successor, do not update their successor predictions to the new observations, as they are most likely noise.

Longer stability periods imply a more stringent low-pass filter. As it turns out, anything more stringent than a simple single-event stability filter is detrimental to overall performance. We demonstrate in the experimental results section



(a) *Update Process*



(b) *Candidate Selection*

Figure 3. Conceptual Views of Noah

below by testing against trace data for varying stability periods. We refer to selecting a successor as the new prediction the second consecutive time it is observed, *i.e.* with stability 1, as "first-pass filtering." First-pass filtering, along with the maintenance of per-file metadata is the main reason for Noah's accuracy in successor predictions.

In this manner, the total metadata that Noah must maintain per file is as little as two file pointers (typically a *long integer*) and even the stability counter described in Figure 3(b) can be discarded as we selected simple first-pass filtering. It is therefore possible to maintain any decision information within the metadata of the current file, and with an efficient implementation overheads are minimal.

This is a small fraction of the space overhead and update requirement that would be required by state-of-the-art predictive caching algorithms with comparable prediction accuracy, such as the Finite Multi-Order Context (FMOC) [6] model, or graph-based models as described by Griffioen and Appleton [2].

3. Experimental Results

To evaluate Noah, the algorithm was tested on file system traces gathered using Carnegie Mellon Univeristy's DFSTrace system [10]. These traces included two main workloads, *barber* a file server, and *mozart* a personal workstation. These traces provide information at the system-call level, and represent the original stream of access events, not filtered through a cache. The main performance metric used was prediction accuracy. If a successor to a particular file is observed to be consistent with what Noah predicted then this is considered to be a successful prediction. The average number of successful predictions for the duration of the trace provides a measure of accuracy which can be considered on a per-file basis, or for summary purpose is averaged over the entire file space.

Averages for both individual file predictor performance, and overall average were calculated, but for our definition of accuracy the overall average is more relevant. This provides a greater weight for more frequently accessed files. This is consistent with the fact that poor predictions on frequent events are more detrimental than poor predictions for infrequent (possibly transient) events. For the two main workloads we consider a trace period of approximately one month, and a longer trace period of a little over a year. Our results are consistent at both time scales and are discussed in more detail below where we compare the performance of Noah against competing models.

3.1. Noah and Optimal Pairings

The ability of Noah to reduce detrimental noise in successor predictions can be seen in the rank-difference plot of Figure 4. This graph is equivalent to Figure 1(a) presented above, with the last-successor model being replaced with Noah. Noah eliminates most instances of the static first-successor model being a better selection. The rank-difference plots for longer periods, and different workloads are very similar.

Figure 5(a) compares the performance of Noah to the last and first successor models. Without resorting to selection between the two models, Noah can consistently outperform both models. These results are for a time period of approximately one moth, but are consistent over trace periods exceeding a full year. Figure 5(b) compares Noah to the optimal pairing algorithm, and in spite of the few negative points in Figure 4, the overall performance of Noah is within 1 to 4% of this optimal selection model.

Figure 4 represented the trace described here as *barber (month)*, which is for file access events on the server *barber* over a period of approximately one month. The results for all other month-long trace periods are consistent, but it is interesting to observe that Noah slightly exceeds the perfor-

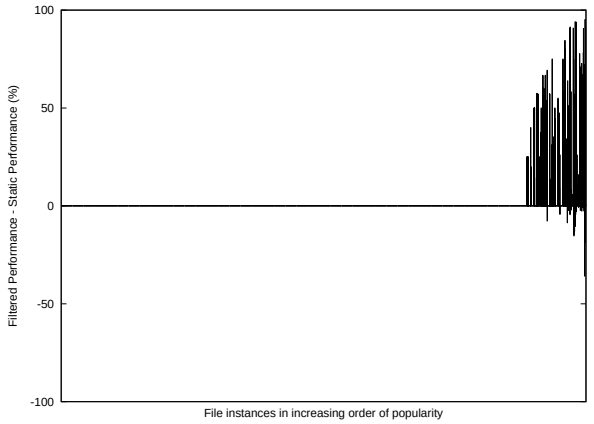
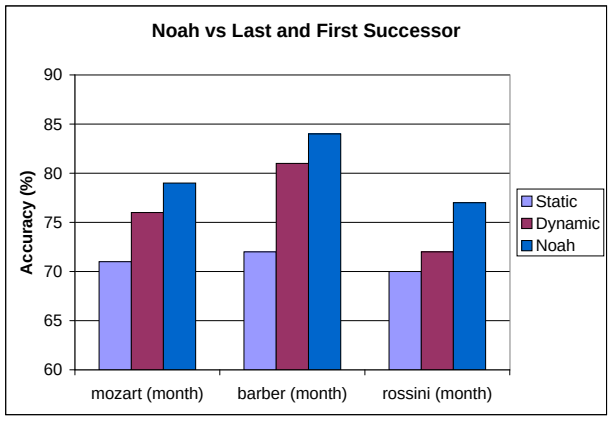
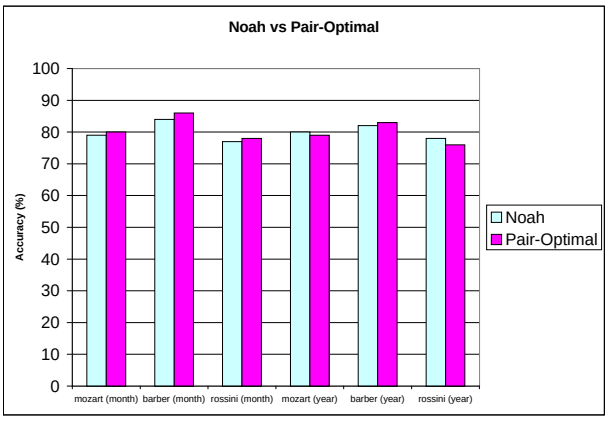


Figure 4. Rank-difference plot for Noah *vs.* the first successor model tested over a period of approximately one month.



(a) Noah *vs.* First and Last Successor Models.



(b) Noah *vs.* the Optimal Adaptive Pairing.

Figure 5. Noah compared to the two single-state successor predictors (dynamic/first-successor and static/last-successor), and an optimal adaptive selection between the two.

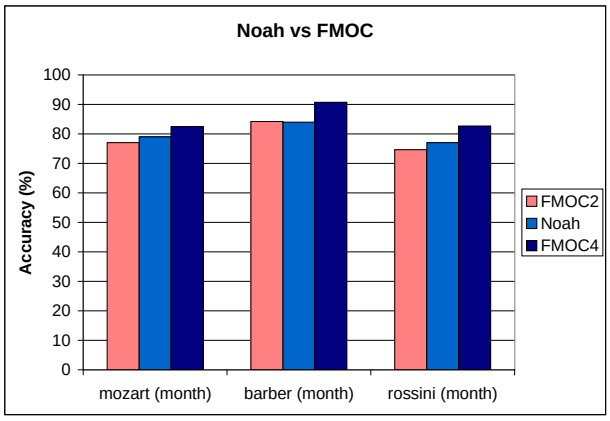
mance of the optimal pairing algorithm for some of the year-long trace periods. The reason for this lies in a reduction of the optimal pairing algorithm's accuracy for lengthier periods of time. Although we have seen that static successor predictions remain valid for extended periods of time, it is only to be expected for their validity to lessen as lengthier time periods are involved, and more high-level shifts in file access behavior occur. Another reason for Noah's excellent performance in this context is its ability to maintain true static pairings. Noah will update a successor prediction if it is observed to remain unchanged for two access events. But should a pairing be strong enough to remain valid for periods exceeding a year, it is highly likely that such a pairing is a strict successor relationship, rarely changing, and therefore rarely requiring updates to Noah's prediction.

3.2. Noah, FMOC, and PCM

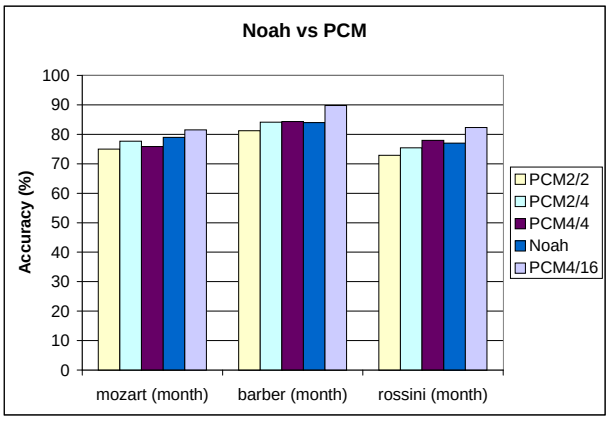
Earlier works on file access prediction include the Finite Multi-Order Context (FMOC) model, and its derivatives the Partitioned Context Model (PCM) and Enhanced PCM (EPCM) [6, 5], which predict based on a context modeling principle inspired from the data compression domain [13]. FMOC was found to generally outperform an earlier technique for file access prediction based on the construction of weighted relationship graphs built from a window of observed successors, first proposed by Griffioen and Appleton [2]. Figure 6(a) compares Noah to the context modeling (FMOC) predictor, and Figure 6(b) compares Noah to PCM with varying amounts of state-space.

An FMOC-n model predicts using a context model of order n, and subsequently requires $O(m^n)$ state space, where m is the number of unique files observed. PCMn/p models a partitioned context model, of order n with state-space limited to a partition size of p for each file. To beat Noah's performance the space-restricted context model (PCM) needs to track up to 16 nodes per file (each with a pointer to another file, and an access count). Although the higher-order FMOC models outperform Noah, they require state space that is polynomial in the number of unique files observed. The simple filtered model of Noah is capable of outperforming PCM models that track the same number of nodes (two), and Noah is capable of matching the performance of models with more than eight times the requirement in state space.

Noah only needs to maintain two predictions with only the dynamic (last-successor) prediction requiring update with each access to a file, as the static prediction remains



(a) Noah *vs.* FMOC.



(b) Noah *vs.* PCM.

Figure 6. Noah performance compared to dedicated prediction algorithms. Noah is compared to the Partitioned Context Model (PCM), with varying order and partition sizes, and to second and fourth-order Finite Multi-Order Context (FMOC) models (with unlimited state space).

unchanged until a replacement is required.

4. Related Work

With increasing network speeds, latency becomes a greater issue than bandwidth for network performance, as the reduction of the problem is limited by the speed of light. The same trends have been observed in disk storage, in the form of seek times being a greater burden despite increases in disk capacity and bandwidth. We predict similar trends in vast storage systems.

To overcome latency problems without some form of prefetching would not appear to be possible, hence the increased interest in prefetching and predictive caching in such applications as web-proxies [11] 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. FMOC was presented in an accuracy comparison with the graph-based model and the last-successor model [6, 8]. It has been used to produce the PCM and Extended PCM prefetching algorithms [5]. The FMOC prediction model is based on context modeling techniques drawn from the data compression domain. The first to suggest the application of techniques from data compression to predictive caching were Curewitz, Krishnan, and Vitter [1, 13].

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]. File hoarding within the Coda project [4] utilized a fairly elaborate derivation of LRU, and yet suffered from excessive dependence on user intervention and setup effort. 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], 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 whether 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 [12]. 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. More recent work on branch prediction [14] maintained dynamic information on a per-branch basis, similar to Noah's per-file metadata. Furthermore they employed a more complex two-level scheme to deal with higher-level workload shifts. 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. Noah assumes

that either the current prediction or a newly established stable successor will be chosen, but in reality the observed successor could potentially be any file in the file system space. The fact that Noah works so well in spite of this observation is indicative of the inherent relatedness in file access events.

5. Conclusions and Future Work

Noah is capable of providing reliable file access predictions utilizing a constant state space, which is a small fraction of more complex prediction algorithms. Noah requires no user-adjustment of operating parameters, and automatically adapts to variations in the observed access patterns of specific files. This can be seen in Noah's usefulness and consistent performance regardless of the time-scale involved. In short Noah is self-tuning, adaptive, accurate, minimal, persistent (per-file information is simple to maintain thanks to the low overheads), and tolerant of noisy or missing observation data (an inability to update the successor predictions will, at worst, reduce Noah to a first-successor model).

The comparison of Noah and predictive caching algorithms should be taken within the context of this paper. It must be realized that as predictors for predicting far into the future, the FMOC and graph-based models are probably superior to the plain Noah implementation. However, Noah makes no attempt to predict far into the future. And yet its consistent accuracy, across multiple files suggests that it can indeed do so, or at least supply useful relationship information for higher-level applications such as file prefetchers and hoarding algorithms. We do not claim Noah to be the most accurate predictor available, but we do propose Noah as a flexible mechanism for establishing highly accurate successor predictions robustly and with incredibly little system overheads. Future work on Noah includes the testing of models that predict multiple successors, as well as applications of these algorithms to data placement and hoarding problems.

6. Acknowledgments

We are grateful to our colleagues in the Computer Systems Laboratory, for their continuous feedback, support and valuable discussions. We are also especially grateful to M. Satyanaryanan of Carnegie Mellon University for providing us with the system traces used in this study. This in turn would not have been possible without the greatly appreciated efforts of T. Kroeger in processing and conversion of these traces.

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)*, pages 257–266, Washington, D. C., May 1993.

[2] J. Griffioen and R. Appleton. Reducing file system latency using a predictive approach. In *USENIX Summer Technical Conference*, pages 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*, C22(11):1025–34, November 1973.

[4] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *Operating Systems Review*, 25(5):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, Mar. 1999. IEEE.

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

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

[9] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–61, 1994.

[10] 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.

[11] 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.

[12] J. E. Smith. A study of branch prediction strategies. In *Proceedings of the Eighth Annual Symposium on Computer Architecture*, pages 135–48, Minneapolis, MN, USA, May 1981. IEEE.

[13] J. S. Vitter and P. Krishnan. Optimal prefetching via data compression. *Journal of the ACM*, 43(5):771–93, Sept. 1996.

[14] T.-Y. Yeh and Y. N. Patt. Alternative implementations of two-level adaptive branch prediction. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 124–34, Gold Coast, Queensland, Australia, May 1992. IEEE.