File Access Prediction with Adjustable Accuracy

Ahmed Amer
a.amer@acm.org
University of California, Santa Cruz

Darrell D. E. Long
darrell@cs.ucsc.edu
University of California, Santa Cruz

Jehan-François Pâris
paris@cs.uh.edu
University of Houston

Randal C. Burns
randal@cs.jhu.edu
Johns Hopkins University

Abstract

We describe a novel on-line file access predictor, recent popularity, capable of rapid adaptation to workload changes while simultaneously predicting more events with greater accuracy than prior efforts. We distinguish the goal of predicting the most events accurately from the goal of offering the most accurate predictions (when declining to offer a prediction is acceptable). For this purpose two distinct measures of accuracy, general and specific accuracy, are presented corresponding to these goals. We describe how our new predictor and an earlier effort, Noah, are capable of trading the number of events predicted for prediction accuracy by modifying simple parameters. When prediction accuracy is strictly more important than the number of predictions offered, trace-based evaluation demonstrates error rates as low as 2%, while offering predictions for more than 60% of all file access events.

1 Introduction and Motivation

File access behavior is not random, it is driven by application program and user behavior. This fact, coupled with the growing performance bottleneck of computer storage systems, has resulted in a significant amount of research into improving file system behavior through predicting future access events. 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.

In predictive prefetching, file access prediction models are usually coupled with a cache to allow the retrieval of data into a high-speed cache before it is actually requested. Predictive prefetching requires the prediction of sequences of file accesses far enough in advance to avoid the predictions being untimely. With bursty requests for data, predicting the next request is not useful if the actual request arrives as quickly as the prediction is available. More recent work [1] has used prediction models to determine inter-file relationships, the cache is modified to utilize such relationship information in fetching preconstructed groups of highly related files. Such a grouping approach avoids the timeliness requirements of predictive caching, and yet is dependent on the quality and nature of the relationship information. It is therefore increasingly important to have efficient file access predictors that are useful for maintaining inter-file relationship information, this information can be used to perform grouping and further-reaching predictions.

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 toward 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, and is a direct application of the models we discuss below. Prior works on mobile file hoarding such as Coda [7] and Seer [10] have made significant strides in automating the process of file hoarding, yet sufficient automation remains an elusive goal.

We revisit the file access predictor, *Noah* [3], and introduce a new predictor, *Recent Popularity*, both of which are very well suited for identifying inter-file relationships. We demonstrate how the definition of prediction accuracy needs to be expanded when predictors are used to estimate inter-file relationships. We go on to demonstrate how *Noah* and *Recent Popularity* can allow the adjustment

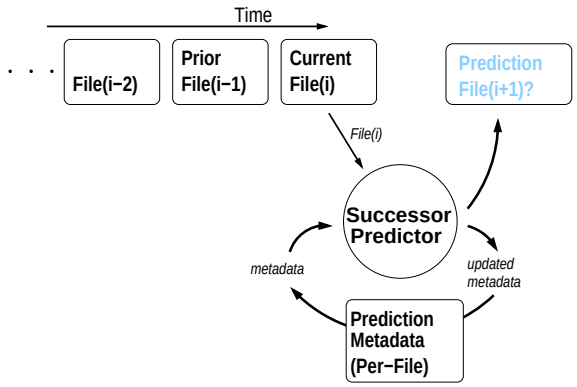


Figure 1: On-line successor prediction.

of their predictions' average accuracy using a simple numeric parameter.

2 Prediction and Predictors

Our goal is to develop an accurate on-line file access predictor that maintains limited per-file state information, and yet can be used as the basis for determining strong inter-file relationships with a high degree of certainty. As illustrated in Figure 1 the role of a predictor in our model is to observe a file access sequence, and for each file access offer a prediction for the following file to be accessed. Such a predictor is allowed to maintain a limited amount of additional metadata, preferably on a per-file basis. We start with a discussion of two baseline models that represent two extremes in adaptability. These baseline models offer a successor prediction for every file that is not being encountered for the first time.[1]

When used to indicate inter-file relationships a predictor should be able to indicate whether it is confident in the prediction it is offering. Our prediction models, *Noah* and *Recent Popularity*, have the ability to decline offering a prediction, or more precisely they are capable of indicating whether or not they have confidence in their prediction. This difference between predictions and guesses indicates a distinction between the number of events correctly predicted, and the number of predictions made correctly. For this purpose we start with a discussion of prediction accuracy in which we define *general* and *specific* prediction accuracy to measure these different goals. We then describe two baseline models for comparison, and go on to describe our predictors which are capable of trading number of predictions made for increased prediction accuracy. These predictors are *Noah* and *Recent Popularity*, both of which can perform this trade-off with the adjustment of a single parameter. *Recent Popularity* has the additional ability to increase the amount of per-file metadata for better performance.

[1] No on-line predictor can be expected to offer predictions for accesses to files that have not been previously encountered.

2.1 Prediction Accuracy

Prediction accuracy is defined as the ratio of events correctly predicted to all events encountered. For file successor prediction, we distinguish two forms of accuracy: *general* and *specific* accuracy. We define general accuracy as the ratio of file access events correctly predicted to the total number of events. Specific accuracy is defined as the ratio of correct file access predictions to the number of predictions offered by the prediction model. This is a particularly important distinction for our predictors, Noah and Recent Popularity, due to their ability to trade improved accuracy for a reduction in the total number of events predicted. For both predictors this trade-off is controlled using a single numeric parameter.

Our predictors do not actually reduce the number of predictions to improve accuracy, they merely reduce the number of predictions that are offered with confidence. They will offer a successor prediction for any event that has been observed at least once in the past, but if greater accuracy is required, then they will offer fewer predictions that satisfy the prediction criteria. The ratio of such "confident" predictions to the number of events will be defined as *prediction coverage*. We shall see in the experimental sections below that it is possible to decrease specific inaccuracy to as low as 2%, while sacrificing less than 40% prediction coverage, which is equivalent to being correct 98% of the time for more than 60% of all file access events.

2.2 Baseline Models

In prior work [2, 3] we have observed that files can be divided into two general 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 can be captured using two simple successor predictors, the last and first successor predictors.

The *Last Successor* policy takes the most recently observed successor of file A as the successor prediction for the next occurrence of A. Lei and Duchamp used such a predictor as a base-line model [11] for comparison against their approach to file prefetching. Kroeger and Long [8] presented a more careful analysis of file access prediction, recognizing a surprisingly high accuracy for this predictor. They noted that when a file prediction scheme is only asked to provide a single candidate for the next event, the last-successor scheme is almost as likely to provide the right candidate as far more elaborate schemes used in predictive prefetching [6, 9, 11].

The *First Successor* policy takes the first observed successor of file A as the successor prediction for all subsequent occurrences of A. Although this may not appear to be a promising policy, file access events are highly related, and both *first* and *last* successor are effective at capturing many inter-file relationships. Whereas last successor could capture changes in inter-file relationships over time, and demonstrated better overall performance, first successor was found to outperform last successor for a substantial fraction of files [2]. This can be understood by considering an access sequence such as:

$$S : ABACABACABACAB...$$

This is an extreme example, for which a last successor predictor would be incorrect for 100% of file A's successors, whereas an unchanging guess would be incorrect only 50% of the time.

2.3 Noah

Noah is a more elaborate predictor which we have presented in prior work [3]. The main innovation of Noah is the extension of the last successor model to filter out noise in the observed access stream, combining the dynamic adaptability of last successor with the prudence of the first successor model.

Noah is based on the intuition that noise is detrimental to the quality of any deductions made from dynamic observations. It ignores observations that vary too rapidly, effectively acting as a low-pass filter. Given an access sequence such as:

$$S : ABABABACABABAB...$$

Last successor would make two prediction errors for the one occurrence of a successor other than B, and the first successor model would be completely incorrect if B was not the successor of A upon its first occurrence. As with the last-successor model, Noah maintains a record of the last file to be a successor to the current file, but it also keeps track of a current successor prediction, which is updated to become the last-successor only if the last successor is observed to have been correct for a *stability* count number of accesses. The required *stability* is a parameter of the Noah predictor. Algorithm 1 illustrates the procedural behavior of Noah successor predictors.

With this *stability* parameter, Noah is able to demand greater certainty of a successor prediction before it is offered as anything more than a good guess. 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 *stability* number of accesses.

Algorithm 1 Predict using *Noah*

```
if LastSuccessor(File(i − 1)) = File(i) then
    Counter(File(i−1)) ← Counter(File(i−1))+1
else
    Counter(File(i − 1)) ← 0
end if
if Counter(File(i)) > stability then
    Prediction(i + 1) ← LastSuccessor(i)
    Predicted ← TRUE
    Guessed ← FALSE
else
    Predicted ← FALSE
    Guessed ← TRUE
end if
```

If this criteria is not satisfied, then the unchanged prediction by Noah is considered a guess, and not a prediction offered with any confidence.

2.4 Recent Popularity (Best j of k)

Our new predictor provides the stability benefits of Noah while allowing for faster adaptation to workload changes. For the list of k most recently observed successors of a file, we select as a prediction the most popular file. We offer this successor as a prediction if it occurs at least j times in this list.

Figure 2 illustrates the operation of recent popularity using an example sequence. The prediction offered by recent popularity with parameters j and k is denoted as $Best_{j\ of\ k}(X|S)$, where S is the observed sequence and X is the current file for which we are making a successor prediction. In the figure we have extracted the lists of immediate successors for each file encountered, and offer example predictions for successors of the file B. Out of the most recent six successors of B the majority is a tie between A and C, each of which occurs three times. The prediction, $Best_{3of6}(B|S)$, is made as A as it occurs more recently than C, recency is the tie-breaker. $Best_{4of6}(B|S)$ offers no prediction with confidence, as the most popular of the last six successors, A, occurs less than four times.

The final example chooses C as the $Best_{4of9}$ prediction, as it occurs the most frequently among the last nine successors of B. By increasing the value of k recent popularity takes a majority of a larger sample. This increases the amount of metadata required per-file, but as we will see below, our experimental results show that larger samples result in better majority-based predictions. Increasing the j parameter makes the prediction criteria more stringent, acting in a similar manner to the *stability* parameter for Noah. Greater j values imply improved specific accuracy and reduced prediction coverage. Recent popularity allows great flexibility in balancing metadata requirements (through adjusting k), specific accuracy and

	File Access Sequence	
S:	A B C D B C D B D B D B C B C B C A B A B A B A B	
	Per-File Successors	*Successor Counts*
A:	B,B,B,B,B	B(5)
B:	C,C,D,D,C,C,C,A,A,A	A(3),C(5),D(2)
C:	D,D,B,B,A	A(1),B(2),D(2)
D:	B,B,B,B	B(4)

$Best_{3of6}(B|S) = A$
$Best_{4of6}(B|S) = \Phi$
$Best_{4of9}(B|S) = C$

Figure 2: Example sequence and corresponding *Recent Popularity* predictions.

prediction coverage (through adjusting j).

3 Empirical Evaluation

Using file system traces we ran experiments to evaluate the general accuracy, prediction coverage, and specific accuracy of our predictors. We compared these results for Noah and recent popularity with the baseline models. We start with experiments that demonstrate the limited variability of file successors. These experiments show that per-file tracking of a very limited number of successors is enough to cover almost all successors that can be predicted by the best possible on-line predictor. For the accuracy and prediction coverage results we will actually plot inaccuracy and omissions. We demonstrate that increasing Noah's stability parameter progressively reduces specific inaccuracy (increases specific accuracy), while simultaneously reducing prediction coverage (increases prediction omissions). We will demonstrate similar behavior for recent popularity, with the added benefit of consistently improved general accuracy.

3.1 Experimental Workloads

Simulations were run on file system traces gathered using Carnegie Mellon University's DFSTrace system [14]. The tests covered five systems for durations ranging from a single day to over a year. The traces represent varied workloads, particularly *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. For clarity we will refer to these traces as *workstation*, *users*, *write*, and *server* respectively. These traces provide information at the system-call level, and represent the original stream of access events not filtered through a cache. For these CMU traces we are measuring the hit-rate for a whole file cache based on file open requests. This assumes a coarse granularity for the analysis,

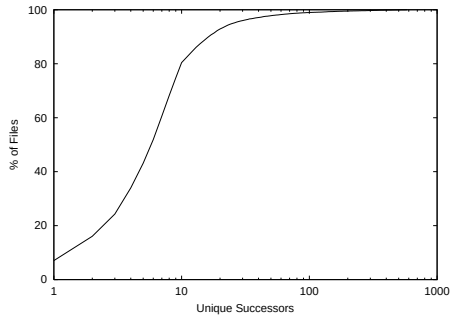


Figure 3: A CDF plot for the *workstation* workload, indicating how many unique successors were observed for all files over a period of one week.

we focus on patterns of file requests and are not concerned with intra-file access patterns.

For a finer level of analysis we also considered individual requests for file data, drawn from the traces provided by Drew Roselli at the University of California, Berkeley [16]. To eliminate any interleaving issues, these UCB traces were processed to represent the workloads of individual workstations, and simulations were run against both instructional and research machines.

3.2 Successor Behavior

Analysis of the workloads demonstrated that file successors show limited variability, and suggest that recency of occurrence is the most consistent and accurate indicator of subsequent occurrence. As an upper bound on how much metadata is needed to track file successors Figure 3 shows a CDF plot of the number of unique successors observed for all files accessed in the CMU *workstation* workload. The data is representative of the other workloads and indicates that over a week, 80% of files have fewer than ten unique successors, and more than 90% of files have less than twenty unique successors. These results are typical over all traces examined for durations ranging from a week to year. For shorter durations (a day or less) there is noticeably less variation, with 90% of files having less than 10 successors. Capturing most inter-file relationships should therefore require a reasonably limited amount of per-file metadata.

The CDF plot of Figure 3 suggests that tracking a relatively small number of successors per file is likely to capture most immediate successors. Figure 4 plots the performance of successor predictions based on per-file immediate successor lists. The *window size* is the the number of unique successors maintained for each file. The y axis represents the miss rate for each scheme. A miss occurs for a scheme if we find that a file's immediate successor was not within the successor list maintained using

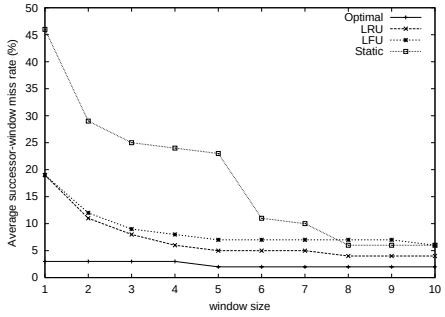


Figure 4: Miss rates for per-file successor windows of varying sizes, and using different replacement policies.

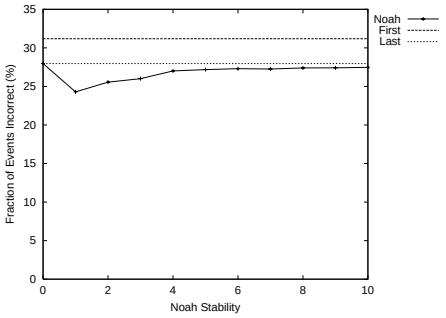
that scheme. Figure 4 presents results for three different update policies, and one upper bound:

- **LRU** – continuously updates the entire per-file immediate successor lists with LRU replacement.
- **LFU** – similar to LRU, but with frequency-based replacement, and recency tie-breakers.
- **Static** – this policy simply tracks the first *window size* unique successors for a file, and never updates the lists when it's full.
- **Optimal** – an oracle that has perfect knowledge of successor events, and will make an accurate prediction if the event to follow has ever occurred previously within *window size* events of the current event. This yields the best performance possible by any on-line algorithm in this application.

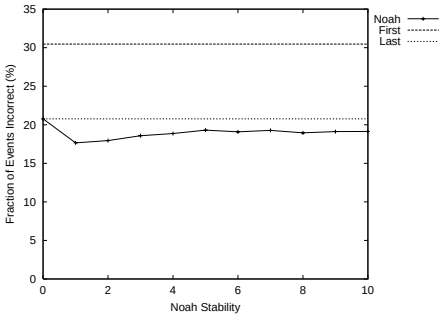
Figure 4 gives an indication of the effectiveness of a limited amount of per-file state in capturing possible successors. The most important result of Figure 4 is how low the miss rates of even the simplest schemes are for window sizes of as few as five successors. With as few as five or six successors, all schemes were generally comparable to the optimal on-line policy. Figure 4 shows that it is possible to capture most likely successors for a file in a very small window, with near-optimal accuracy, and utilizing minimal metadata updates. These results support the claim that successor variability is generally low, and the consistent superiority of LRU suggests that recency may be the best indication of a successor's future importance.

3.3 The Accuracy of Noah

Figure 5 shows the general inaccuracy of Noah compared to the first and last successor models. The x axis represents increasing values for stability from zero to ten. It should be noted that when Noah uses a stability value of zero its behavior is equivalent to that of the last successor model. The y axis represents the percentage of all file access events not correctly predicted or guessed by the



(a) *workstation*

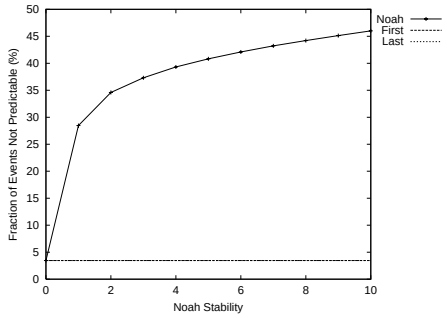


(b) *server*

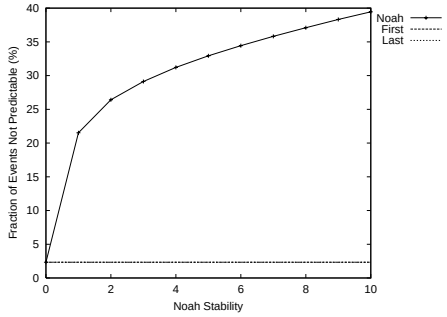
Figure 5: Noah's General Inaccuracy for increasing stability values.

predictors. Last and first successor are indicated as constant values, as they have no parameters and are only affected by the workload. For general accuracy, the best performance of Noah is found with a stability value of one. This means the most events can be predicted correctly if we employ a last successor policy that waits for one further access before changing its successor prediction. The performance gain for the more adaptable last successor model is more pronounced for the *server* workload than the *workstation* workload, yet for both workloads we see a further 3 to 5% performance improvement when using Noah. In this and most following figures we only present results for the *workstation* and *server* workloads, but they are representative of all workloads tested.

In Figure 6, the x axis represents Noah's stability, while on the y axis we plot the percentage of events for which Noah and the baseline models could not offer a confident prediction. The baseline models always offer a confident prediction when a prediction is possible. The prediction omissions of 2 to 3% (prediction coverage of 97 to 98%) plotted for the baseline models represent the percentage of file access events that presented the predictors with a new file, never encountered before. As expected for Noah, the percentage of omitted predictions increases as the re-

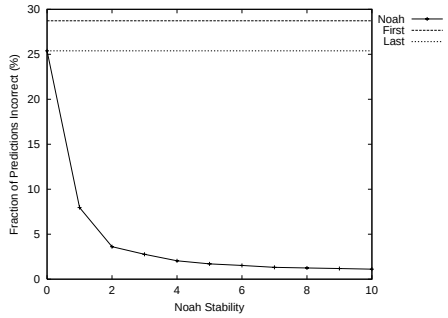


(a) *workstation*

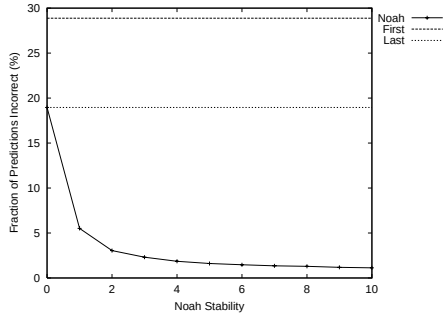


(b) *server*

Figure 6: Noah's Prediction Coverage for increasing stability values.



(a) *workstation*



(b) *server*

Figure 7: Noah's Specific Inaccuracy for increasing stability values.

quired stability is increased.

Figure 6 indicated that increasing the required stability to ten increased the percentage of predictions not made to the range of 40 to 50%. This may seem like a substantial price to pay, but by considering Figure 7 we see what this reduction in predictions offers in return. Figure 7 plots the specific inaccuracy of Noah compared to the baseline models. The x axis again represents stability values. The most impressive thing to note on this graph is the rapid reduction in specific inaccuracy. With stability values starting around four, we quickly see Noah's specific prediction inaccuracy reduced to as little as 2 to 4%. This means that 96 to 98% of all predictions offered by Noah are correct. Furthermore, from the prediction coverage results (Figure 6) we know that with a stability value of four Noah offers confident predictions for almost 70% of all file access events.

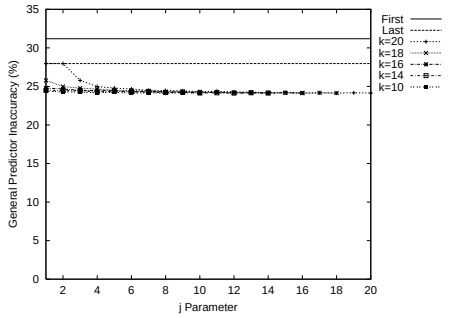
In short, by increasing the stability parameter of Noah we are able to reduce specific prediction inaccuracy to as low as 2% while still offering predictions for more than two thirds of all access events encountered. Noah's failing appears to be in the general inaccuracy results, for which we do not see a consistent behavior for increased stability values.

3.4 The Accuracy of Recent Popularity

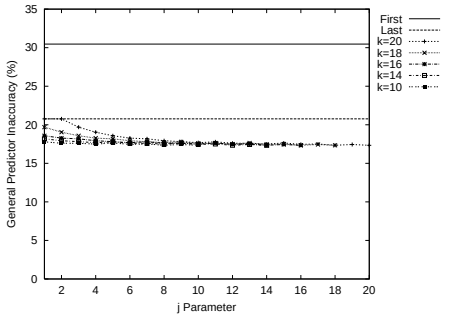
Noah's problem of non-decreasing general inaccuracy is not faced when using recent popularity, which shows more monotonic general inaccuracy trends. Recent popularity simultaneously allows greater control over the trade-off with prediction coverage.

Figure 8 shows the general inaccuracy of recent popularity compared to the first and last successor models. The x axis represents increasing values for the required majority parameter (the j parameter). The y axis represents the percentage of all file access events not correctly predicted or guessed by the predictors. Last and first successor are again indicated as constant values, while for recent popularity we provide multiple plots for values of k ranging from ten to twenty. Plots for recent popularity are of varying length because j values cannot exceed k values. In other words, you cannot require a majority greater than the number of successors maintained, the quorum cannot be greater than the total number of members.

The most important feature of Figure 8 is how consistent the reduction in general inaccuracy is for all variations in the parameters of recent popularity. As with Noah, there is a reduction in total incorrectly predicted events by approximately five percent. Unlike Noah, this

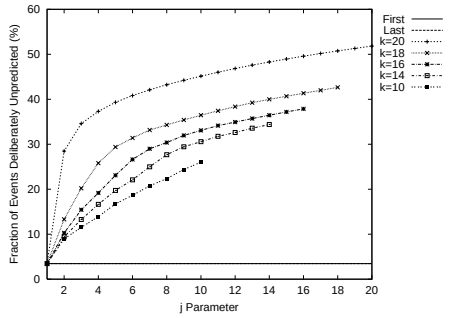


(a) *workstation*

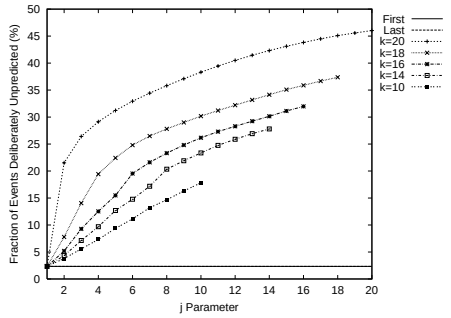


(b) *server*

Figure 8: Recent Popularity's General Inaccuracy for increasing j values and different k values.



(a) *workstation*



(b) *server*

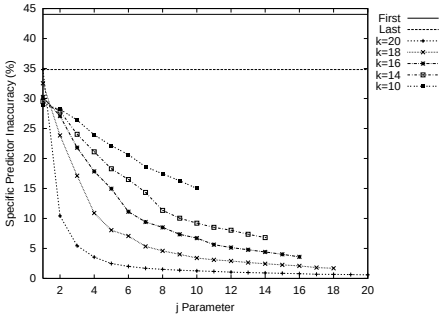
Figure 9: Recent Popularity's Prediction Coverage for increasing j values and different k values.

reduction does not exhibit a local minimum, but is consistent for increasing values of j (the required majority). We have presented only the *workstation* and *server* workloads, but this behavior is consistent across all tested workloads.

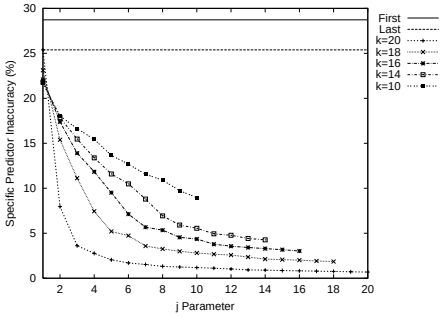
Figure 9 shows the effect of increasing the j parameter on prediction coverage. As with Noah's stability parameter, increasing the required majority j results in a greater number of events not resulting in a confident prediction. Unlike Noah, this reduction is much less pronounced when shorter per-file successor lists (smaller k parameters) are used. When a list of only ten prior successors is maintained, the prediction omissions do not exceed 20 to 25%, giving a prediction coverage of 75 to 80%.

For improving general accuracy, it would appear good to choose lower j and k values, where recent popularity could provide the greatest prediction coverage using the least metadata, and yet still provide good prediction improvement. Maintaining longer per-file successor lists (increased k values) provides a benefit when we consider the specific accuracy of recent popularity. Figure 10 plots the specific inaccuracy of recent popularity compared to the baseline models. We show results for all four tested CMU workloads and Figure 11 shows similar results for sample instructional and research systems from the Berkeley workloads. Between the Berkeley and CMU workloads there is a noticeable difference in the performance of the baseline models, specifically the first successor model performs very poorly compared to last successor. The point we wish to illustrate is the consistency in the behavior of recent popularity across all workloads. We see a steady decrease in specific inaccuracy as we increase the j parameter. Furthermore, this decrease is more pronounced as we increase the k parameter.

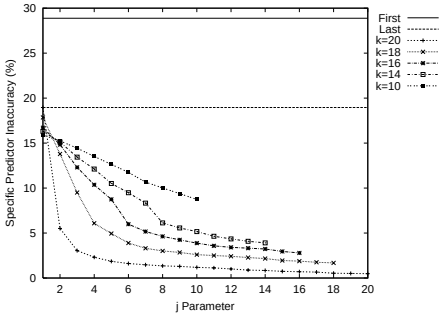
With recent popularity, the greater the number of per-file successors maintained (the k parameter), the more pronounced and rapid the effect of increasing the required majority (the j parameter). If greater general prediction accuracy is required, shorter successor lists with lower majority requirements (low j and k values) result in a consistent and reliable improvement in general accuracy, while requiring less additional metadata and metadata updates. If we wish to use recent popularity for accurate relationship estimation we can trade reduced prediction coverage for increased specific accuracy by increasing the j value. If we want to increase the effect of increasing the j value on specific accuracy, we can increase the k values (thereby sacrificing additional metadata).



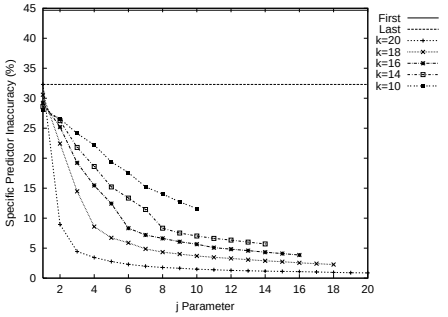
(a) *users*



(b) *workstation*

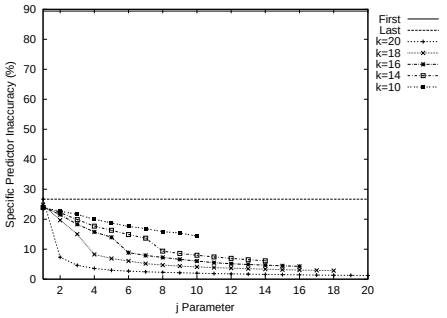


(c) *server*

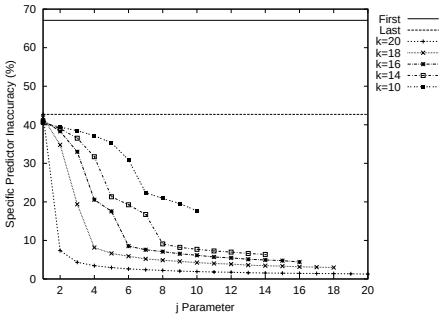


(d) *write*

Figure 10: Recent Popularity's Specific Inaccuracy for increasing j values and different k values.



(a) *instructional workstation*



(b) *research workstation*

Figure 11: Recent Popularity's Specific Inaccuracy for the Berkeley traces.

4 Related Work

Our work has drawn from prior work in distributed file systems, predictive prefetching, and data grouping. Griffioen and Appleton presented a file prefetching scheme based on graph-based relationships [6]. Their probability graphs are limited to tracking frequency of access within a particular "look-ahead" window size. On the other hand, our predictors are primarily based on immediate recency (succession), Our models are also independent of any concept of look-ahead window size, as we limit our predictors to the task of predicting successors. Recent work by Shriver *et al.* [18] has provided analytical reasoning for the benefits of read-ahead buffering and prefetching. The use of the last successor model for file prediction, and more elaborate techniques based on pattern matching, were first presented by Lei and Duchamp [11]. Later work by Kroeger and Long [8] introduced the additive accuracy metric, which weighted prediction accuracy by the numeric prediction likelihood offered by the predictor. Our measure of specific accuracy is similar, but requires much less information from the predictor. Specific accuracy requires an indication of confidence or lack thereof in the prediction, additive accuracy requires a numeric estimate

of likelihood. Kroeger and Long's work also compared the predictive performance of the last successor model to Griffioen and Appleton's scheme, and more effective schemes based on context modeling and data compression [9]. The first proposed application of data compression techniques to file access prediction was presented by Vitter and Krishnan [4, 25]. Prior works on Noah [3] have compared it to Kroeger and Long's [8] predictors. There is also a significant body of work on using transparent compiler-directed approaches [13] and application-level hints for improved prefetching [15].

Grouping has been applied for data placement. The earliest such works attempted used frequency-based estimates of access likelihood to optimize the placement of popular data. Attempts to optimally place files on disk were originally done manually, placing frequently accessed files closer to the center of the disk. The need to automate this process was addressed by the work of Staelin and Garcia-Molina [20–22]. This work dealt with optimal placement, but offered models based on the assumption that file access events are independent. These approaches made no attempt to capture dynamic relationships between files. The Berkeley Fast File System (FFS) [12, 19] includes attempts to group related data, *e.g.* file data and metadata, into cylinder tracks on disk.

Dynamic groups [23] attempt to exploit inter-file relationships, but require explicit application hints to determine group membership. Earlier work on the automatic detection of working sets includes the work of Tait and Duchamp [24]. The *Seer* project also attempted to use file groups, but for mobile file hoarding [10]. Seer used a relationship estimator based on the overlap of file open and close events, and applied a clustering algorithm to build file hoards from such related files. In our approach we are not dependent on such a specific measures of inter-file relationship, and make no attempt to construct a large file hoard. Instead, we require only knowledge of the sequence of file access events, and determine arbitrarily accurate inter-file relationships. Examples of the state of the art in automated file grouping include C-FFS [5] (collocating FFS), which bases grouping on a directory-membership heuristic, and Hummingbird [17] which utilizes the underlying structure of web files. In contrast, our model does not require any knowledge of underlying data structures, as our predictors establish succession predictions based on observed file access behavior, as opposed to inference from file location or content.

5 Conclusion

We have defined general and specific accuracy as distinct measures of prediction accuracy. The former indicating total events correctly predicted and useful for predictors used in prefetching, while the latter indicates the accuracy of predictions offered by a prediction model that can decline to make a prediction if its criteria are not satisfied. We presented two predictors with such criteria, Noah with a simple stability parameter, and the new Recent Popularity predictor. Both predictors are capable of reducing the number of predictions made in exchange for an increased likelihood of those predictions being accurate. Recent popularity shows more consistent improvements in general accuracy than Noah, and the additional ability to trade metadata requirements for greater sensitivity to the prediction criteria.

When prediction accuracy is strictly more important than the number of predictions offered, trace-based evaluation has demonstrates error rates as low as 2%, while offering predictions for more than 60% of all file access events. Predictions for more events can be made at the expense of reductions in accuracy, and for both Noah and recent popularity this trade-off is controlled through one simple numeric parameter.

6 Acknowledgements

We are especially grateful to Thomas Kroeger for valuable feedback, support and discussions. We are grateful to all the members of the Computer Systems Laboratory, for their continuous feedback, support and valuable discussions. Our most extensive multi-year traces were kindly made available by M. Satyanaryanan of Carnegie Mellon University, through the greatly appreciated efforts of Thomas Kroeger in processing and conversion. We are also grateful to Drew Roselli for making the UCB traces available.

References

[1] A. Amer and D. D. E. Long, "Aggregating caches: A mechanism for implicit file prefetching," in *Proceedings of the Ninth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2001)*, (Cincinnati, OH), pp. 293–301, IEEE, Aug. 2001.

[2] A. Amer and D. D. E. Long, "Dynamic relationships and the persistence of pairings," in *Proceedings of the International Workshop on Wireless Networks and Mobile Computing (WNMC 2001)*, (Mesa, Arizona), pp. 502–7, IEEE, Apr. 2001.

[3] A. Amer and D. D. E. Long, "Noah: Low-cost file access prediction through pairs," in *Proceedings of 20th International Performance, Computing,*

and Communications Conference (IPCCC 2001), (Phoenix, Arizona), pp. 27–33, IEEE, Apr. 2001.

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

[5] G. R. Ganger and M. F. Kaashoek, "Embedded inodes and explicit grouping: Exploiting disk bandwidth for small files," in *Proceedings of the 1997 USENIX Annual Technical Conference*, (Anaheim, CA), pp. 1–17, Jan. 1997.

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

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

[8] 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), pp. 14–9, IEEE, Mar. 1999.

[9] 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*, (Boston, MA), June 2001.

[10] G. H. Kuenning and G. J. Popek, "Automated hoarding for mobile computers," in *16th ACM Symposium on Operating Systems Principles*, (Saint Malo, France), pp. 264–75, Oct. 1997.

[11] H. Lei and D. Duchamp, "An analytical approach to file prefetching," in *Proceedings of the 1997 USENIX Annual Technical Conference*, (Anaheim, CA), pp. 275–88, Jan. 1997.

[12] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry, "A fast file system for UNIX," *ACM Transactions on Computer Systems*, vol. 2, pp. 181–97, Aug. 1984.

[13] T. C. Mowry, A. K. Demke, and O. Krieger, "Automatic compiler-inserted I/O prefetching for out-of-core applications," in *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation (OSDI)*, (Seattle, WA), pp. 3–17, Oct. 1996.

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

[15] 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)*, (Copper Mountain Resort, CO), pp. 79–95, Dec. 1995.

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

[17] E. Shriver, E. Gabber, L. Huang, and C. Stein, "Storage management for web proxies," in *Proceedings of the 2001 USENIX Annual Technical Conference*, (Boston, MA), pp. 203–16, June 2001.

[18] E. Shriver, C. Small, and K. Smith, "Why does file system prefetching work?," in *Proceedings of the 1999 USENIX Annual Technical Conference*, (Monterey, CA), pp. 71–83, June 1999.

[19] K. A. Smith and M. Seltzer, "A comparison of FFS disk allocation policies," in *Proceedings of the 1996 USENIX Technical Conference*, (San Diego, CA), pp. 15–25, Jan. 1996.

[20] C. Staelin and H. Garcia-Molina, "Clustering active disk data to improve disk performance," Tech. Rep. CS-TR-283-90, Department of Computer Science, Princeton University, Feb. 1990. revised June 1990.

[21] C. Staelin and H. Garcia-Molina, "File system design using large memories," in *Proceedings of the Fifth Jerusalem Conference on Information Technology (JCIT)*, pp. 11–21, IEEE, Oct. 1990.

[22] C. Staelin and H. Garcia-Molina, "Smart filesystems," in *Proceedings of the Winter 1991 USENIX conference*, pp. 45–51, Jan. 1991.

[23] D. C. Steere, *Using Dynamic Sets to Reduce the Aggregate Latency of Data Access*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Jan. 1997.

[24] C. D. Tait and D. Duchamp, "Detection and exploitation of file working sets," Tech. Rep. CUCS-050-90, Computer Science Department, Columbia University, New York, NY 10027, 1990.

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