Title

The case for efficient file access pattern modeling

Authors

Kroeger, TM

Long, DDE

Publication Date

1999

DOI

10.1109/hotos.1999.798371

The Case for Efficient File Access Pattern Modeling

Thomas M. Kroeger and Darrell D. E. Long[†‡]
Jack Baskin School of Engineering
University of California, Santa Cruz

Abstract

Most modern I/O systems treat each file access independently. However, events in a computer system are driven by programs. Thus, accesses to files occur in consistent patterns and are by no means independent. The result is that modern I/O systems ignore useful information.

Using traces of file system activity we show that file accesses are strongly correlated with preceding accesses. In fact, a simple last-successor model (one that predicts each file access will be followed by the same file that followed the last time it was accessed) successfully predicted the next file 72% of the time. We examine the ability of two previously proposed models for file access prediction in comparison to this baseline model and see a stark contrast in accuracy and high overheads in state space. We then enhance one of these models to address the issues of model space requirements. This new model is able to improve an additional 10% on the accuracy of the last-successor model, while working within a state space that is within a constant factor (relative to the number of files) of the last-successor model. While this work was motivated by the use of file relationships for I/O prefetching, information regarding the likelihood of file access patterns has several other uses such as disk layout and file clustering for disconnected operation.

1 Introduction

Many I/O systems benefit extensively from sequential prefetching. For example, disk controllers frequently do *read-ahead*, prefetching of the next disk block. File systems often prefetch the next sequential block in a file. In both of these cases, prefetching can be done because there is a concept of the next sequential data element intrinsic in the abstraction. However, there is no intrinsic concept of the next file in a sequence. This limits the ability of file systems to prefetch data across files.

File accesses follow previous patterns with a high probability [3, 5, 7]. Using traces of file system activity, we demonstrate the extent of these relationships, by observing that a simple *last-successor* prediction model (which predicts that an access to file *A* will be followed by the same file that followed the last access to *A*) is able to correctly predict 72% of access events.

We then examine two models previously presented for learning file relationships by tracking access patterns. After each event both of these models provide a list of predicted files to be accessed next and a probability associated with each file. If the next file accessed was one of those predicted, then that model's score is increased by the probability with which that file was predicted. These scores are then normalized over the number of total events in each trace, providing a measure we call *additive accuracy*, that ranges from zero to one.

The first model [3] examined is based on a graph, that keeps frequency counts for all files accessed within a window following an access to each file. The second model [5] was adapted from the text compression technique *finite multi-order context modeling* (*FMOC*), to track file access patterns. It uses a tree to keep frequency counts of access patterns, tracking a finite number of multiple-order patterns (order in this case means the length of the pattern tracked). The primary difference between these two models is that the *graph-based* model considers subsequent accesses as an unordered set (a window) to allow for interleaving of accesses patterns from different sources, while the *FMOC* tracks the order in which subsequent accesses occur. The *graph-based* model sees extremely poor *additive accuracy*, 0.400 on average as compared to the *last-successor*'s accuracy of 0.720. The success of the *last-successor* indicates that there would be little interleaving of different patterns in the traces used, and helps to explain the poor performance of the *graph-based* model. The *FMOC* at 0.818 sees a significant improvement over the *last-successor* (reducing the difference between *last-successor* and an ideal predictor by greater than one third), but at the cost of significantly greater overhead.

We then present an improved *FMOC* model called a *Partitioned Context Model* [6] (*PCM*). This model addresses the key limitation of model state space. We show that this efficient model sees a marginal improvement in accuracy

[†] Internet: tmk,darrell@cse.ucsc.edu, Telephone (831) 459-4458.

[‡] Supported in part by the National Science Foundation under Grant CCR-9704347, the Usenix Association and Office of Naval Research under Grant N00014–92–J–1807.

while working within a state space that is within a constant factor of the *last-successor* model (typically 8 to 16 nodes per file instead of one node per file). The *PCM* model sees an *additive accuracy* of 0.820.

This work is motivated by the use of the above models to improve I/O system caching decisions. However, models that accurately estimate the access relationships between files have significant value for many other applications. Placing related information in close proximity on disk has been shown to improve I/O performance dramatically [2]. Additionally, the grouping of files that are likely to be accessed within close temporal proximity is a problem faced by systems that try to hoard files for disconnected operation as well as lay out data for tertiary storage systems (*e.g.* a tape robot putting related files on the same tape).

The rest of this paper is organized as follows: §2 presents the *last-successor*, *graph-based* and *FMOC* models, §3 presents the trace data used and the results of our tests with these three models, §4 introduces *Partitioned Context Modeling* and presents the results from its predictions. We then discuss related work in §5, future work in §6 and conclude in §7.

2 File access prediction

We present three models for predicting file accesses. The first, is a simple *last-successor* model that serves as a baseline. This model predicts that an access to file *A* will be followed by the same file that followed the last access to *A*. This model requires only one node per unique file within the traces so we can say that its state space is $O(n)$, where n is the number of unique files. The second model was originally proposed by Griffioen and Appleton [3] and uses a probability graph that keeps counts of accesses within each node and uses weighted edges to indicate previous access proximity. The last [5] is derived from the text compression field and uses a tree with access counts in each node.

2.1 Graph based modeling

Griffioen and Appleton first proposed modeling file access patterns with a probability graph that keeps frequency counts of accesses that follow within a window of a specified length. This model maintains a graph where each node represents a distinct file. When file A is accessed the count in its node is increased. Then for a window of subsequent accesses (*e.g.* file B C and D) an edge with a count of 1 is made connecting file A's node to the nodes representing these files. If an edge connecting the files already exists then the count for this edge is incremented. The unordered nature of the window of accesses is intended to allow the interleaving of access streams from different activities. This model keeps one node for each file, but for each of these nodes must also track the count on each edge, so we can say that the state space for this model is $O(n^2)$. Figure 1 shows a sample probability graph.

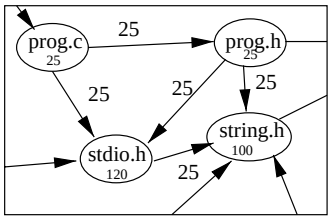

Figure 1: An example probability graph

2.2 Context modeling

Finite Multi-Order Context models originate from the text compression algorithm PPM [1]. A context model is one that uses preceding events to model the next event. For example, in the string "**object**" the character "**t**" is said to occur within the context "**objec**". The length of a context is termed its *order*. In the example string, "**jec**" would be considered a third order context for "**t**". Techniques that predict using multiple contexts of varying orders (*e.g.*"**ec**", "**jec**", "**bjec**") are termed *Multi-Order Context Models* [1]. To prevent the model from quickly growing beyond available resources, most implementations of a multi-order context model limit the highest order modeled to some finite number m, hence the term *Finite Multi-Order Context Model*. In the examples here we have used letters of the alphabet to illustrate how this modeling works in text compression. For modeling file access patterns, each of these letters is replaced with the name of a unique file.

A context model uses a *trie*, a data structure based on a tree, to efficiently store sequences of symbols. Each node in this trie contains a symbol (*e.g.* a letter from the alphabet, or the name of a specific file). By listing the symbols contained on the path from the root to any individual node, each node represents an observed pattern. The children of every node, represent all the symbols that have been seen to follow the pattern represented by the parent.

To model access probabilities we add to each node a count of the number of times that pattern has been seen. Since the children of a node represent all the symbols that have previously followed that node's sequence, then the sum of their counts should equal the count of that node. The one exception to this case is when the node represents an event that has just occurred and the model has not had a chance to see what event will follow. In this case, the frequency count is equal to the sum of its children's counts plus one. Therefore, we can use the formula $count_{child}/(count_{parent} - 1)$ to estimate the likelihood of a child's symbol occurring as the next event.

Figure 2 extends an example from Bell [1] to illustrate how this trie would develop when given the sequence of events *CACBCAABCA*. In this diagram the circled node *A*

represents the pattern *CA*, which has occurred three times. This pattern has been followed once by another access to the file *A* and once by an access to the file *C*. The third time is the last event to be seen and we haven't yet seen what will follow. We can use this information to predict both *A* and *C* each with a likelihood of 0.5. The state space for this model is proportional to the number of nodes in this tree, which is bounded by $O(n^m)$, where m is the highest order tracked and n is number of unique files. This large state space results because this model treats the following events as an ordered (instead of unordered) sequence. In §3 we see that for the traces we examine, tracking this ordering is important to the accuracy of a model.

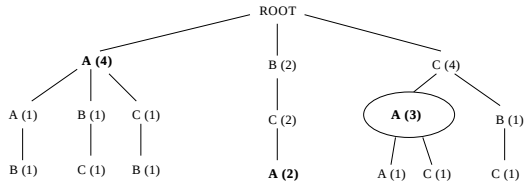

Figure 2: Example trie for the sequence *CACBCAABCA*.

3 Experimental results

To study inter-file access patterns of a computer system, we used trace data collected with the DFSTrace system. This system was used by the Coda project [13] to trace all system calls on 33 machines over a period ranging from February of 1991 to March of 1993. For this work we selected the workload of four specific machines for the month of January 1993. These machines were selected to represent a diverse set of workload characteristics. The machine *barber* was a server with the highest rate of system calls per second. The machine *dvorak* had the highest percentage of write activity, *ives* had the largest number of users, and *mozart* was selected as a typical desktop work station. Table 1 provides some summary statistics of these trace sets. The column marked *rate* indicates the rate of system calls. The column marked *comp miss* represents the compulsory misses or, the fraction of accesses where the file accessed has not been previously accessed (number of unique files divided by the number of events). Such events are not predictable by any online model. So, one minus this fraction can be used to represent a bound on the *additive accuracy* measure of our models.

Table 1: Trace data summary information (Length is in hours, rate is in calls per second).

Machine	Length	Rate	# Records	Comp M	bound
ives	591	4.18	8886861	0.0179	0.982
barber	554	16.42	32764738	0.0210	0.979
dvorak	760	5.19	14203240	0.0287	0.971
mozart	511	2.93	5390288	0.0436	0.956

In order to measure and compare the accuracy of these models we define the measure *additive accuracy* as follows: after each file access if the file accessed was among those predicted by the model, then the model's accuracy for that event is the likelihood with which this file was predicted. To measure a model's accuracy with respect to a specific sequence of accesses (*e.g.* a trace of file activity), we consider each successful prediction independently and sum these values. This number is then normalized over the total number of events in the sequence, to produce an *additive accuracy* measure. This measure is a number between 0 and 1 which represents how well a model was able to predict the next event.

In the case of the *last-successor* model only one file was predicted after each access, so each correct prediction increased the score by one. Thus the *additive accuracy* for a *last successor* model is also the fraction of times that the model was able to correctly predict the next event. Figure 3 shows the results of our tests for the *last-successor* model, a *graph based* model for windows of size two (Graph 2) and four (Graph 4), and a *FMOC* model for the second (*FMOC* 2) and fourth (*FMOC* 4) order.

The *additive accuracy* scores for the *graph-based* model are significantly lower than those of the *FMOC* model. We note that the *graph-based* model's performance decreases as window size is increased, while for the *FMOC* model the accuracy increases as model order is increased. Additionally, when comparing the results across the four different workloads we see that there is little difference. One might expect that the workload with the highest call rate or greatest number of users might have greater interleaving and benefit more from the unordered window that the graph models use. This is not the case, *barber* with the highest workload is slightly greater for all models. *Ives* with the greatest number of users is not distinctly different than any of the other traces. Based on these observations and the success of the *last-successor* model it is apparent that there is little interleaving between various predictive sequences. Therefore, the need for the access window that the *graph-based* model uses does not exist in these traces.

Additionally, as of this writing we are in the process of running the same tests on a set of traces taken recently at Berkeley [12]. These traces represent three workloads, research, instructional and web server, with activity rates ranging from 23 to 107 calls per second. Preliminary results from these traces confirm the results seen with the Coda traces, with no distinct difference between the various workloads.

In addition to accuracy our tests also examined the model space used by each of these models. In §2 we presented the bounds on the state space of these models. Figure 4 shows the actual number of nodes required for each model, normalized over the total number of records in each trace. For the *graph-based* model we count one node for each edge and one for each file. The *FMOC* model space is simply

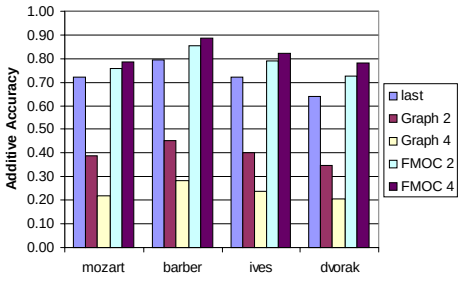

Figure 3: *Additive Accuracy* scores

the number of nodes in the trie, and for the last-successor we count one node for each unique file accessed. This figure shows that both models scale poorly in comparison to the *last-successor*. Additionally, the *FMOC* with its bound of $O(n^m)$ requires significantly more space than the *graph-based* model.

Based on the results shown here, we are able to make two conclusions for the traces studies. First, significant information is lost by disregarding the order in which events occur. Second, the *last-successor* model shows that file reference patterns can be accurately modeled with a constant amount of information for each file. In §4 we present a modified *FMOC* that addresses several issues of scale and is able to track higher order sequences in the state space that is within a constant factor of the *last successor* model with respect to the number of files.

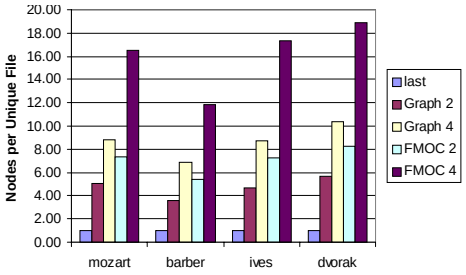

Figure 4: Model space requirements (normalized over the number of unique files)

4 Partitioned context modeling

We addressed the issue of model state space by modifying the *FMOC* model presented in §2. This technique, *Partitioned Context Modeling* [6], partitions the trie based on first order nodes and then limits the size of each partition. We tested this model on our trace data and saw that not only was it able to work efficiently in a state space that was within a constant factor of the *last-successor* model, but also that the prediction accuracy was improved because of its ability to adapt faster to changes in the access patterns.

Each first order node in the trie of a *FMOC* model represents an access pattern of length one, consisting of one access to the file represented by that node. Since removing any first order node would result in the loss of all the information associated with patterns that begin with the file represented by that node, we pursued an approach to limit the model size by not purging any first order nodes but instead limiting their descendants.

This approach divides the trie into partitions, where each partition consists of a first order node and all of its descendants. The number of nodes in each partition is limited to a static number that is a parameter of the model. The effect of these changes is to reduce the model space requirements from $O(n^m)$ to $O(n)$. Figure 5 shows the trie from Figure 2 with these static partitions.

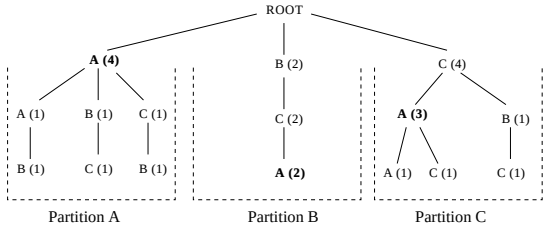

Figure 5: Example partitioned trie for the access sequence *CACBCAABCA*.

When a new node is needed in a partition that is full, all node counts in the partition are divided by two (integer division), any nodes with a count of zero are cleared to make space for new nodes. If no space becomes available, the access is ignored. Another benefit of restricting space in this manner is that when new access patterns occur, existing node counts see this exponential decay, causing the model to adapt faster to new access patterns.

4.1 Partitioned context model results

To test the accuracy and adaption of the *Partitioned Context Model*, we ran it through the trace data for a model order ranging from one to four and a partition size ranging from four to 1024. We saw that a PCM required significantly less space than and was marginally more accurate than *FMOC*, averaging an *additive accuracy* of 0.820. Since the *PCM* maintains less state, this can only be attributed to the model's increased adaptability.

To examine how the model's partition size affected *additive accuracy*, we graphed prediction accuracy over variations in partition size. Figure 6 shows a typical graph of the effects of partition size on the prediction accuracy. From this graph we can see that a small amount of state space, eight to 12 nodes per file, will provide enough information to represent nearly all of the predictive information about access patterns beginning with that file. Additionally, as the partition size increases the model becomes less adaptive to changes and we see a marginal decline in the performance.

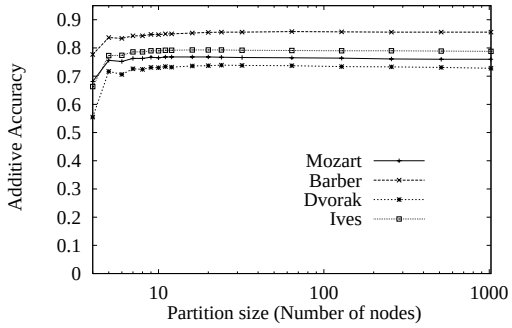

Figure 6: Partition size versus *Additive Accuracy* for a second order *PCM*. (Note the X axis is log scale ranging from four to 1024)

Figures 7 and 8 compare the prediction accuracy results for the *PCM*, *last-successor* and second and fourth order *FMOC*. *PCM* 2/8 and 4/32 represents a second order model with a partition size of eight and a fourth order model with a partition size of 32. These figures show that the *PCM* is able to provide prediction accuracy equal to that of a *FMOC*, while working in a state space of the same order as *last-successor*. In practice a typical partition would take approximately 348 bytes, with 12 bytes per node (eight for a unique ID, two for a count and two for child and sibling pointers). Although this is significantly greater than the eight bytes per file required for a *last-successor* model, the *PCM* model reduces by one third the amount of inaccuracy of the *last-successor* model. With a space cost of 384 bytes per file it is quite reasonable to envision an *i-node* like structure for each file that indicates its relationships to other files.

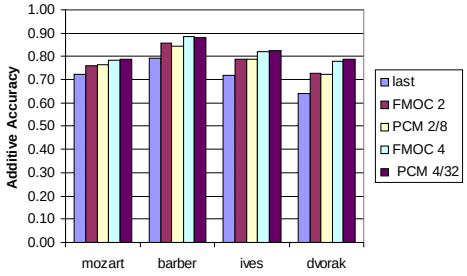

Figure 7: *PCM*'s prediction accuracy results.

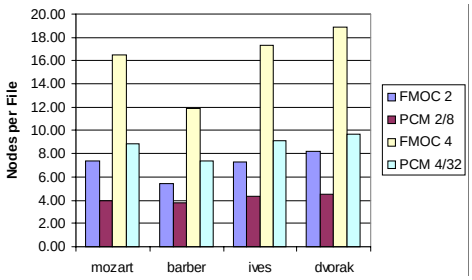

Figure 8: *PCM* space sizes.

4.2 Analysis of PCM

To better understand why this partition model is so effective, we consider the worst case for context models. This case is one in which the access frequencies of all the children of a selected file (A) are evenly distributed. In such a case all events following an access to A would be independent of each other, however this work is based on the inter-dependence between files. By statically limiting the number of data points used in the partition for file A, we are requiring that the majority of the distribution's probability mass is able to be kept within this many data points. If this is not the case then an access to file A is not strongly correlated with accesses to any other files, and it has little predictive information to offer.

5 Related work

The use of compression modeling techniques to track access patterns and prefetch data was first examined by Vitter, Krishnan and Curewitz [14]. They proved that for a Markov source such techniques converge to an optimal online algorithm, and go on to test this work for memory access patterns in an object-oriented database and a CAD System. Chen *et al.* [4] examine the use of *FMOC* type models for use in branch prediction. Griffioen and Appleton [3] were the first to propose this *graph-based* model that has seen use across several other applications [11, 10]. Lei and Duchamp [8] have pursued modifying a UNIX file system to monitor a process's use of fork and exec to build a tree that represents the processes execution and access patterns. Kuenning *et al.* [7] have developed the concept of a *semantic distance*, and used this to determine groupings of files that should be kept on local disks for mobile computers. Madhyastha *et al.* [9] used hidden Markov models and neural networks to classify I/O access patterns within a file.

6 Future work

This work has started several discussions into better metrics for the accuracy of each model. We intend to rigorously formalize the problem space, and apply several other measures. Some of the measures that have been discussed are: an entropy based measure $G = \sum -\log(p_i)$ where p_i is the probability with which the occurred event was predicted; a squared loss function $L = \sum(1 - p_i)^2$ and a zero-one measure where only the item with the highest probability is predicted. Additionally, a threshold based measure, that models prefetching based on a *probability threshold* has also been discussed. As mentioned in §3 we have begun using additional traces taken recently by the Now project at UC Berkeley.

The partitioned model presented here is one successful method for efficiently restricting finite multi-order context

models. Other variations of this model still require further exploration. We intend to explore many extensions of this model, for example, including recency and frequency in the likelihood estimates, and removing the order limitation on the partitions. This work has chosen to examine these predictive models separate from an application, we intend to use these models for cache prefetching within the Coda filesystem.

7 Conclusions

For the traces examined, there is a strong degree of correlation between file accesses. While there is no intrinsic concept of the next sequential file there do exist probabilistic relationships between files. This information is of great value not only for file system prefetching, but also for disk layout and related file grouping in general.

We have demonstrated that while a simple *last successor* model can do quite well to predict file relationships, a more intelligent model that adapts to changing access patterns can do even better within the same order of space constraints. Finally, we have shown that tracking ordering and linear model space are critical components of any effective file access pattern model. We believe that models like the *partitioned context model* that maintain a fixed amount of predictive information and adapt as patterns change can provide significant improvements for I/O system performance.

Acknowledgments

We are grateful to Prof. Satyanarayanan and the Coda/Odyssey group for all their kind assistance with the trace data. Richard Golding, Randal Burns, David Kulp, and David Helmbold for their discussions on methods to measure the model's accuracy. The members of the Concurrent Systems Laboratory and Rebecca Vega have been of significant help. We are grateful to these and the many other people who offered their time and support.

References

[1] T. C. Bell, J. G. Cleary, and I. H. Witten, *Text Compression*. Englewood Cliffs, New Jersey: Prentice Hall, 1990.

[2] G. R. Ganger and M. F. Kaashoek, "Embedded inodes and explicit grouping: Exploiting disk bandwidth for small files," in *1997 Annual Technical Conference, January 6–10, 1997. Anaheim, CA* (USENIX, ed.), (Berkeley, CA, USA), pp. 1–17, USENIX, Jan. 1997.

[3] J. Griffioen and R. Appleton, "Performance measurements of automatic prefetching," in *Parallel and Distributed Computing Systems*, pp. 165–170, IEEE, September 1995.

[4] T. N. M. I-Cheng K. Chen, John T. Coffey, "Analysis of branch prediction via data compression," in *The Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 128–13, SIGOPS, ACM, October 1996.

[5] T. M. Kroeger and D. D. E. Long, "Predicting file-system actions from prior events," in *Proceedings of the USENIX 1996 Annual Technical Conference*, USENIX, January 1996.

[6] T. M. Kroeger, "Predicting file-system actions from prior events," Master's thesis, University of California, at Santa Cruz, 1997.

[7] G. Kuenning, "The design of the seer predictive caching system," in *Workshop on Mobile Computing Systems and Applications*, pp. 37–43, IEEE Computer Society, December 1994.

[8] H. Lei and D. Duchamp, "An analytical approach to file prefetching," in *Proceedings of USENIX 1997 annual Technical Conference*, USENIX, January 1997.

[9] T. M. Madhyastha and D. A. Reed, "Input/output access pattern classification using hidden Markov models," in *Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems*, (San Jose, CA), pp. 57–67, ACM Press, Nov. 1997.

[10] J. N. Matthews, D. Roselli, A. M. Costello, R. Y. Wang, and T. E. Anderson, "Improving the performance of log-structured file systems with adaptive methods," in *Proceedings of the 16th Symposium on Operating Systems Principles (SOSP-97)*, vol. 31,5 of *Operating Systems Review*, (New York), pp. 238–251, ACM, Oct.5–8 1997.

[11] V. N. Padmanabhan and J. C. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency," in *Proceedings of the 1996 SIGCOMM*, ACM, July 1996.

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

[13] M. Satyanarayanan, "Coda: A highly available file system for a distributed workstation environment," in *Second IEEE Workshop on Workstation Operating Systems*, (Pacific Grove, CA, USA), Sept. 1989.

[14] J. S. Vitter and P. Krishnan, "Optimal prefetching via data compression," *Journal of the ACM*, vol. 43, pp. 771–793, September 1996.