IADIS International Journal on Computer Science and Information Systems
Vol. 4, No. 2, pp. 28-40
ISSN: 1646-3692

FAST FLASH MEMORY CACHING BASED ON FILE ACCESS FREQUENCY

ChenHan Liao. *Building 52, AMAC, SOE, Cranfield University, Bedfordshire, England, MK43 OAL.*
chenhanliao@googlemail.com

Frank Wang. *Building 52, AMAC, SOE, Cranfield University, Bedfordshire, England, MK43 OAL.*
f.wang@cranfield.ac.uk

Na Helian. *School of Computer Science University of Hertfordshire, Hatfield, AL10 9AB, UK.*
n.helian@herts.ac.uk

Sining Wu. *Building 52, AMAC, SOE, Cranfield University, Bedfordshire, England, MK43 OAL.*
s.wu@cranfield.ac.uk

YuHui Deng. *Building 52, AMAC, SOE, Cranfield University, Bedfordshire, England, MK43 OAL.*
y.h.deng@cranfield.ac.uk

ABSTRACT

In modern file systems, traces monitor the file operations and user behaviors. Inevitably, large amount of data continuously produced in daily manner. We show that the knowledge hidden behind system traces can help us understand the system and user behaviors. In this paper, we illustrate that once a file is created with a set of attributes, such as name, type, permission mode, owner and owner group, its future access frequency is predictable. A decision-tree-based predictive model is established to predict whether a file will be frequently accessed or not. By consulting with the rules generated from the predictive model over diverse real-system NFS traces, the model can predict a newly created file's future access frequency with sufficient accuracy. We further introduce an evolutionary storage system, which employs the predicted frequency information to decide what files to keep in a fast storage device, flash memory. The trace-driven experimental results indicate that the performance speedup due to the prediction-enabled optimization is 2-4 compared with base case.

KEYWORDS

Storage system, Access Frequency, Prediction, KDD

1. INTRODUCTION

File system designers have suggested that the knowledge behind system behaviors and file attributes such as file access pattern, size, and name can conduct the improvement and the design of file caching policies and disk layout. The recent researches (T. M. Kroeger & D. D. E. Long, 1999) (Daniel Ellard, et al., 2003) have illustrated that file size, lifespan and access mode (write, read, etc.) are predictable to improve file pre-fetching and caching. Unfortunately, no work has been reported in predicting file access frequency. Since the file access frequency can directly reflect a file's usage and decide whether or not the file is hot, therefore, the storage system may use the file access frequency information to improve its performance. In this paper, we show that applications already give useful hints to a file system, in the form of file attributes at creation time, and that the file system can successfully predict its future access frequencies from these hints (Fig.1). The file access frequency in our work is defined as the frequency of various file operations, which include read, write, execute, lookup and rename.

During the training Period, a predictor for file frequency is constructed from observations of file system activity in trace. The file system can then use this model to predict the frequency of newly-created files. In this paper, we first show that the future file access frequency is predictable and strongly related to the file attributes at creation time. Then, we construct a decision tree based predictive model to infer a new file's future frequency class (high/hot, low/cold). Finally, we present an evolutionary storage system integrating the above predictor. The rest of the paper is organized as follows: Section 2 discusses the related works. Section 3 describes and analyses three NFS traces used in our analysis and the statistical evidence of file access frequency prediction. Section 4 presents FFP (File Frequency Predictor). Section 5 validates FFP. Section 6 presents an evolutionary storage system integrating FFP. Section 7 concludes.

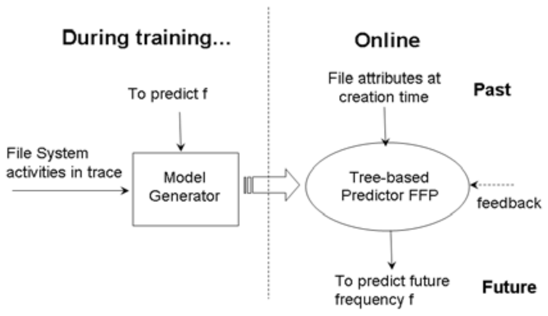


Figure 1. Decision tree-based predictor FFP is a bridge linking the past (attributes at creation) and the future (frequency)

2. RELATED WORKS

Since the gap of processing speed between I/O and CPU performance increases continuously, many efforts have been devoted to address it through optimizing various aspects of file system. One of the solutions is to reduce the number of disk requests by improving the caching efficiency. Another typical solution is to reduce the disk requesting time through re-organizing the disk layout of the relative files' blocks. Based on these solutions, Fast File System (FFS) (Pei Cao, et al., 1996) was designed to handle files in different manners regarding to their size. It places small files on disk so that they are near their metadata. Furthermore, it assumes that files in the same directory are most likely accessed together in a short period.

In addition to file size, other properties such as access type: write-mostly or read-mostly, have been confirmed useful to design various file system policies. For example, Log-structured File System (LFS) proposed by John K. Ousterhout and Fred Douglis, treats the disk as a circular log and writes sequentially to the head of the log. The design of log-structured file systems is based on the hypothesis that write latency is the bottle neck of modern file systems. As a result, a Hybrid file system structure designed against the nature of read and write has been found useful to form high performance file system (Keith Muller, 1997).

Besides above heuristics, it is noticed that file access frequency is also a significant factor influencing the storage system performance. A data file that is currently being heavily accessed by users is a high frequency file. In a Hierarchical Storage Management system (HSM), file I/O may be constantly monitored in order to migrate the high frequency files to the fastest storage devices or to internal memory for better performance. HSM was first implemented by IBM on their mainframe computers (IBM, 2006) to reduce the cost of data storage, and to simplify the retrieval of data from slower media. The user would not need to know where the data was stored and how to get it back, the computer would retrieve the data automatically. The only difference to the user was the speed at which data is returned.

Based on our investigation, the problem of adopting file access frequency is that all these heuristics are embedded into file system it selves, which means if the heuristics are wrong or dynamic the system performance will degrade (Daniel Ellard, et al., 2003). In addition, file system workloads may change over time, the designed policies may take critical time to adapt to it. In previous work, a solution to this problem is that enable applications to supply hints to the file systems (Pei Cao, et al., 1996) (R. Hugo Patterson, et al., 1995). Unfortunately, this solution requires the modification of applications, which hasn't been widely deployed.

In fact, the workloads, user behaviors and applications already provide very rich hints to conduct file system behaviors. An attributes based file properties prediction method has been introduced recently (Daniel Ellard, et al., 2003). It utilizes machine learning approach to analysis the previous file system behaviors and then makes decisions advising file systems the estimated future properties of a new file, such as size, life span and even write-mostly or read-mostly.

However, to the best of our knowledge, there is no reported investigation against file access frequency prediction so far. In this work, we will show that once a file is created with a set of attributes, such as name, type, permission mode, owner and owner group, its future access frequency is predictable. A decision tree-based predictive model will then be established. The predicted frequency information will be used to decide what files to keep in a fast storage device, flash memory. Hence the work is believed to be timely and novel considering file access frequency is useful to realize most of the performance gains.

3. THE TRACES

In order to obtain particular rules and patterns from file system, we collected system traces carrying all necessary file I/O information for our analysis. The Traces we used were taken from three servers that have different users and behaviors.

"DEAS03 traces a Network Appliance Filer that serves the home directories for professors, graduate students, and staff of the Harvard University Division of Engineering and Applied Sciences. This trace captures a mix of research and development, administrative, and email traffic. The DEAS03 trace begins at midnight on 2/17/2003 and ends on 3/2/2003." (Daniel Ellard, et al., 2003) "EECS03 traces a Network Appliance Filer that serves the home directories for some of the professor, graduate students, and staff of the electrical Engineering and Computer Science department of the Harvard University Division of Engineering and Applied sciences. This trace captures the canonical engineering workstation workload. The EECS03 trace begins at midnight on 2/17/2003 and ends on 3/2/2003." (Daniel Ellard, et al., 2003) "CAMPUS traces one of 14 file systems that hold home directories for the Harvard College and Harvard Graduate School of Arts and Sciences (GSAS) students and staff. The CAMPUS workload is almost entirely email. The CAMPUS trace begins at midnight 10/15/2001 and ends on 10/28/2003." (Daniel Ellard, et al., 2003)

For the privacy issue, some information of these three traces was encoded by using the method described in (Gregory R. Ganger & M. Frans Kaashoek, 1997). The encoded traces contain anonymized UIDs, GIDs, and IP address remapped to the new values. There are no information lost among those identifiers and other variables. During the anonymization UIDs, GIDs, and host IP numbers are simply remapped to new values, so no information is lost about the relationship between these identifiers and other variables in the data. All three traces were collected with nfsdump (Daniel Ellard & Margo Seltzer, 2003).

Table 1. The average percentage of read, write, lookup, get attributes, and access operations for a fourteen day trace.

Host	Read	Write	Lookup	Getattr	access
DEAS03	48.7%	15.7%	3.4%	29.2%	1.4%
EECS03	24.3%	12.3%	27.0%	3.2%	20.0%
CAMPUS	64.5%	21.3%	5.8%	2.3%	2.9%

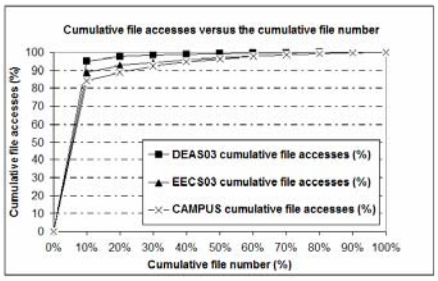


Figure 2. The cumulative file accesses versus the cumulative file number, both expressed as percentages.

Table 1 gives a summary of the average hourly operation counts and mixes for the workloads captured in the traces. These show that there are differences between these workloads, at least in terms of the operation mix. CAMPUS is dominated by reads and more than 85% of the operations are either reads or writes. DEAS03 has proportionally fewer reads and writes and more metadata requests (getattr, lookup, and access) than CAMPUS, but reads are still the most common operation. On EECS03, meta-data operations comprise the majority of the workload.

It was found that more than eighty percent of the file accesses are probably going to less than ten percent of the files. In Fig.2, we plot the cumulative file accesses versus the cumulative file number, both expressed as percentages. The file accesses in our work include read, write, execute, lookup and rename operations. The curve of DEAS03 in Fig.2 shows 94.9% of the file accesses going to 10% of the files. This result with curve of EECS03 and CAMPUS become less skewed. In EECS03 88.9% of the file accesses are going to less than 10% of the total files whereas in CAMPUS 84.1% of the file accesses are going to less than 10% of the total files. CAMPUS is similar to EECS03, but contains a lot of sequential access patterns. We can see that some file accesses are highly skewed and experience shows that only a small fraction of files in a file system is actively and frequently used. As also shown in Fig.2, the exact shape of the file request distribution varies from workload to workload.

In other words, large amount of file I/O are absorbed by only small portion of files. The question is that how to identify whether a certain file is going to absorb large amount of I/O, that is, whether or not this file is going to be highly accessed once it is created. The fact is that file attributes are strongly related to its future access frequency. For example, the lecturers regularly update his/her lecture notes in the file server. Consequently, in the next short period, the files created with UID=”professor A” will be accessed frequently. In contrast, for example, if a student submits his/her coursework to the file server, in the next short period, the files created with UID=”student B” will not hold a good probability of being accessed as frequent as previous one regarding to the popularity reason.

Similarly to this, file mode can also provide useful information to file access frequency. According to the observation (, any files with a mode of 777 in DEAS03 trace is likely to live for less than one second and contain zero bytes of data. It suggests that if a file is created with a mode that makes it readable and writeable by any user on the system is actually never read or written by anyone. For example, lock files (files that are used as a semaphore for interpocess communication) usually exists only for indicating that a process desires exclusive access to a particular file. Therefore, the information hidden behind file access mode can also be used to evaluate for file access frequency.

To identify the associations between the create-time attributes of a file and its future access frequency, we first scan the traces to obtain the previous knowledge and calculate the file access frequency. In these 3 NFS traces, the attribute “fid” uniquely identifies a file; therefore, we can record the access frequency of a file by accumulating the number of occurrences of a certain “fid”. Once we obtain the access frequency of a file in a short period, say peak hours (10am-1pm), we are able to measure the statistical association between each attribute and the relative file access frequency.

To statistically confirm the degree that how strong an attribute of a file relates to the file’s access frequency, we use Chi-square test (James T. McClave, et al., 1997) to roughly analyze the association between high access frequency files and each attribute. Chi-square test lets us know the degree of confidence we can have in accepting or rejecting a hypothesis. Concretely, the hypothesis of our Chi-square test is whether or not a high access frequency file’s attributes and its access frequency class (high) are associated enough. Given access frequency boundaries are 599, 19, and 17 respectively regarding to the 10% top frequent accessed files out of the total files, the Fig.3 shows the confidence of the high access frequency class supported by each attribute over three NFS traces.

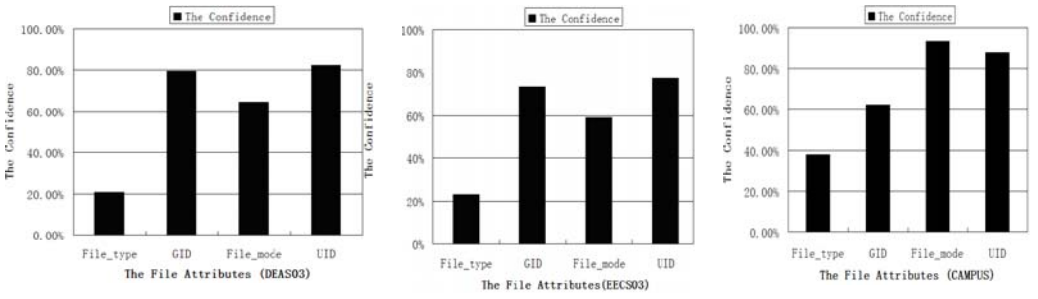


Figure 3. The confidence of file access frequency of attributes

The figures above reveal the confidence of file access frequency supported by the diverse attributes. It can be seen from the figures that the confidence values supported by each attribute are different in each trace. However, some attributes such as UID always show great confidence to the file access frequency, which is one of the representative cases we have discussed at the beginning of this section.

As the Chi-square test is a rough statistical analysis of the confidence of file access frequency, there is a need to precisely evaluate the degree of confidence supported by each attributes. Here we borrow the concept from information theory, Information Gain, to discover how much contribution an attribute can provide to the confidence of file access frequency prediction. This approach is also used to build a more accurate statistical analysis model, decision tree. In the next section, we discuss our decision-tree based predictive model.

4. FILE ACCESS FREQUENCY CLASSIFIER (FFP) CONSTRUCTION

The Chi-square test in previous section has provided us the evidence that files attributes are strongly related to files access frequency. It suggests us that if we know a file’s attributes, we then can predict a file’s frequency class (high or low) based on a predefined frequency boundary. Apart from those representative attributes mentioned above, some other attributes such as the file’s creation time, file name and suffix may also support file access frequency. In this paper, we focus on 4 attributes: file mode, file type, user ID, and group ID.

In UNIX file system, user ID and group ID identify whom the file belongs to and its group. Correspondingly, file mode claims the permission title in three separate parts. For example, if a file contains a user ID “john”, group ID “G1” and the file mode “-rw-r-r”, then it claims that “john” is the owner of this file and has the group “G1”; in the file mode, first set value “rw” denotes that “john” can read and write this file, second set value “r” claims that group users belongs to “G1” can only read this file; the third set value denotes other users can only read this file. In addition, UNIX file system normally contains 4 main file types: ordinary files, directory files, device files, and link files.

To accurately evaluate the confidence given by files' attributes, we use decision tree as our predictive model. We first obtain training sample to build the predictive model, and then validate the model with new files to check whether or not the predictions are accurate. There are many learning algorithm for data classification, one the most popular algorithms is Iterative Dichotomiser 3 (ID3) decision tree, which utilizes the information gain to evaluate the confidence supported by an attribute. Compared with other supervised learning models, decision tree has a few advantages that are of benefits to our problem. First of all, decision tree is simple to understand and interpret. Due to the change of storage system behaviors, system administrator can easily capture the trends of future file access based on the current decision tree's layout. Secondly, decision tree model can generate important insights describing a situation (its alternatives, probabilities, and costs) and their preferences for outcomes. This information makes the predicting results more reliable. In addition, decision tree is a white box model. If a given result is provided by a model, the explanation for the result is easily replicated by simple math. (An example of a black box model is an artificial neural network since the explanation for the results can be excessively complex for a decision maker to comprehend.) To the decision tree model, categorical attributes and fixed classes are required. For the first requirement, since file attributes such as file type, file mode, UID, GID are all symbolic represented with no inherent ordering, therefore our classification problem satisfies this requirement. For the second requirement, we can define a caching performance-based frequency boundary to distinguish low frequency and high frequency.

To determine the frequency boundary between High & Low, the capacity of the caching devices is considered. Since our implementation of FFP is based on a flash memory-oriented storage system which stores frequently used files in flash memory for caching and prefetching, the boundary of file access frequency is determined in such a way that the amount of I/O caused by accessing High frequency files should fit in the maximal capacity of the flash memory. In other words, the boundary is determined by the flash memory capacity. However, how much percentage of the storage space should flash memory occupies? As shown in figure2 of section 3, the average workloads data skewness is 90%. In fact, various workloads may have different data skew. For example, in our implementation, according to the workloads data skewness, we choose the flash memory with capacity equal to 10% of the hard disk storage capacity. That is, if hard disk is 10 GB, then flash memory is 1GB. Consequently, the file access frequency boundary is determined accordingly.

The files we collected are from three NFS traces: DEAS03, EECS03 and CAMPUS. Through scanning the traces, we can capture the file attributes (file type, file mode, UID, GID) and its access frequency by accumulating the occurrences of the file requests. The following table shows a sample taken from DEAS03, the frequency is regarding to one hour (03/02/2003, 00:00-01:00). According to the average data skewness in these three traces, 599 is set to the frequency boundary between low frequency and high frequency of the sample from DEAS03. Then, we can obtain the frequency class in accordance of their real access frequency.

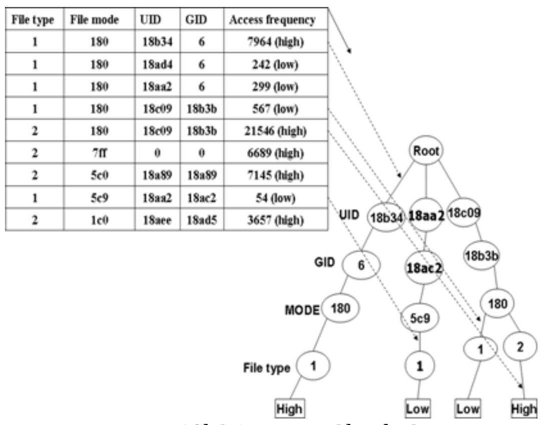
Table 2. Training sample from DEAS03

Fill type	Fill mode	UID	GID	Access frequency
1	180	18b34	6	7964(high
1	180	18ad2	6	242(low)
1	180	18aa2	6	299(low)
1	180	18c09	18b3b	567(low)

2	180	18c09	18b3b	21546(high)
2	7ff	0	0	6689(high)
2	5c0	18a89	18a89	7145(high)
1	5c9	18aa2	18ac2	54(low)
2	1c0	18aee	18ad5	3657(high)

By a given training sample, ID3 algorithm takes all attributes and count their information gain, then choose the attribute for which information gain is maximum, at last make nodes containing that attribute. From the observation of ID3 algorithm, we noticed that from the top of the tree to the bottom of the tree, the upper level nodes hold larger information gain, which means the upper level nodes that belong to a certain attribute make more contribution for classifying the access frequency. The Fig.6 is the ID3 decision tree built regarding to the above training sample.

It can be seen from Fig.4 that begin from the Root, each branch represents a classification rule and its class label. In Fig.6, not all rules are populated in the example tree so that the example decision tree is more readable.



File type	File mode	UID	GID	Access frequency
1	180	18b34	6	7964 (high)
1	180	18ad4	6	242 (low)
1	180	18aa2	6	299 (low)
1	180	18c09	18b3b	567 (low)
2	180	18c09	18b3b	21546 (high)
2	7ff	0	0	6689 (high)
2	5c0	18a89	18a89	7145 (high)
1	5c9	18aa2	18ac2	54 (low)
2	1c0	18aee	18ad5	3657 (high)

IF UID=18b34 THEN Check GID;
IF GID=6 THEN Check File_mode;
IF File_mode=180 THEN Check File_type;
IF File type=1 Then frequency class=high

Figure 4. ID3 decision tree of the training sample.

In addition, as we have discussed that each branch represents a classification rule, therefore, we then can obtain a Boolean representation of the rule. Thus, for a newly created file, it is very simple to judge its frequency class. One of the branches above can be represented as Boolean rule. The classifier we built in this section is based on the decision tree classification algorithm ID3. However, the training sample we used in the example for building the classifier is very small. Generally, more training data used for building the classifier more precise the result reaches the reality. In next section, we will discuss the validation of FFP.

5. THE VALIDATION OF FFP

For other time-irrelevant classification problems, training samples and test samples are normally taken from same workloads to guarantee that both samples have same data characteristics. However, in our problem, those traces were fetched in an uninterrupted time series over several days. Thus, each hour’s trace may differ from each other in terms of file access frequency. It is impossible to split the whole time series workload into first half and second half to build classifier and validate it. As we can imagine, in peak hours, file access frequency could be much higher than rest of hours.

To evaluate the accuracy of FFP over the time series traces, we built the classifiers based on the peak hours’ traces (10am-1pm) on Monday (02/17/2003) as the training sample. Then we predict the files created during the peak hours in the following days to check whether or not the precision is acceptable. The following tables show the accuracy of the predictions during the peak hours on Tuesday, Wednesday and Thursday.

Table 3. Prediction accuracy of FFP on Tuesday

Frequency Class	DEAS03	EECS03	CAMPUS
High	90.1%	92.5%	93.3%
low	98.2%	91.2%	90.6%

Table 4. Prediction accuracy of FFP on Wednesday

Frequency Class	DEAS03	EECS03	CAMPUS
High	88.4%	88.0%	90.1%
low	93.6%	89.6%	87.7%

Table 5. Prediction accuracy of FFP on Thursday

Frequency Class	DEAS03	EECS03	CAMPUS
High	82.6%	87.7%	84.1%
low	88.6%	85.3%	82.2%

Fig.5 shows the accuracy curve of FFP from 02/17/2003-03/12/2003. Although the accuracy of FFP slightly decreases, the overall predicting accuracy remains steady during a reasonable long time.

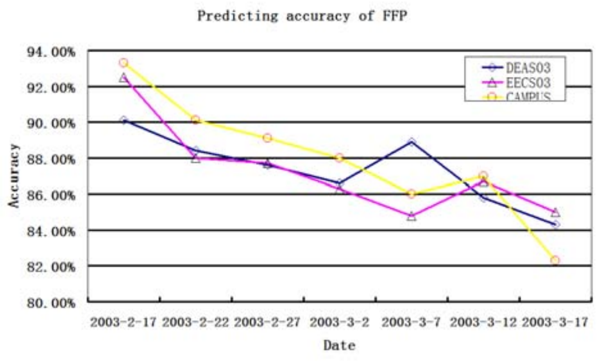


Figure 5. Predicting accuracy of FFP over 3 NFS traces from 2003-2-17 to 2003-3-17

The validation results show that the accuracy of the classifier decreases after few days. This is caused by the slightly changed user behaviors and files' life span. For instance, some lecture notes might be accessed frequently in the next few days after the corresponding lectures were given. However, those lecture notes' access frequency may be no longer frequent once some new lectures are given and their relative lecture notes are created. Hence, there is a need to build a new predictive model to adapt the file system change. The possible way is to rebuild the predictive model when the precision of the model becomes unacceptable.

6. AN EVOLUTIONARY STORAGE SYSTEM INTEGRATING FFP

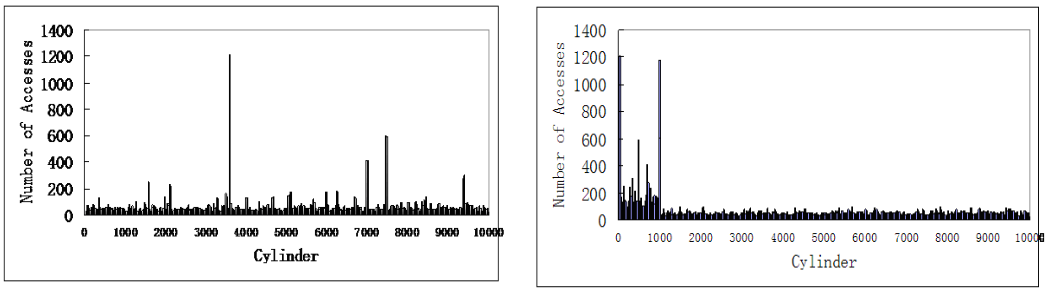
This case study describes a deployed Evolutionary Storage System (EvoStor) with measurable benefits. It is a practical product with an embedded File Frequency Predictor (FFP).

Degradation in performance over time is a serious and common concern in magnetic disks. Sometimes, people benchmark their disks when new, and then many months later, and are surprised to find that the disk is getting slower. In fact, the disk most likely has not changed at all, but the second benchmark may have been run on tracks closer to the center of the disk (Frank Wang, 2006). To counteract the above disk devolution process, we propose to develop a FFP-plugged evolutionary storage system that is evolving over time.

In Section 3, we found that more than eighty percent of the file accesses are going to less than ten percent of the files. If the blocks storing hot (frequently accessed) files are spread over the surface of the disk, distant from each other, long seek delays may result. According to Amdahl's law, we need only allocate a small percent of the data (files) to a fast device such as a flash memory according to the predicted frequencies by FFP to realize most of those performance gains. The EvoStor system combines NAND-based Flash with rotating storage media. The ultra-high-density benefits of magnetic storage technology are preserved, while the ultra-low-power, ultra-high-reliability and fast read/write access of advanced NAND technology enhances the overall value of the EvoStor system. Most traces exhibit a great deal of skewness in file request distribution and therefore the fast flash memory can be made very small (in this experimental setup it is 10% of the disk capacity) without severely impacting the performance of an adaptive disk.

The allocation may be imperfect -- files need only be near their optimal location. The hope of placing frequent files close to each other is that this will significantly reduce the disk seek time between accesses: If it is likely that consecutive accesses are to hot files, then it is likely that they will be to files in the flash memory, reducing the expected disk seek time. The disk rotational delays may also be reduced, if it is likelier that accesses will be to the same flash memory. As we will see, the improvements due to both effects can indeed be very significant.

Fig.9 shows a typical cylinder access distribution over 25,000 requests on a Quantum Atlas disk. In most environments, disk accesses display significant spatial and temporal locality. Indeed, as the narrowness of the peak in Fig.6 (a) shows, only a small number of cylinders are accessed with any significant frequency. Fig.6 (b) shows the effect of allocating the most frequently-accessed files predicted by FFP together into the flash memory with a capacity of 10% of the disk space.



(a) As measured on the original layout;

(b) After the predicted frequency is used to decide what files to keep in a flash memory (10% of the total space).

Figure 6. The predicted frequency information could be used to decide what files to keep in a flash memory. Measured on a Quantum Atlas disk

The Evolutionary Storage system allocates frequent files predicted by FFP to a flash memory. This is a “White Box” or “Gray Box” design in which a predictor passes semantic information of file frequency with a prior probability to a storage system when the file is created. Unlike the above, a “black box” design infers statistically object frequency with a posterior probability in a disk drive, which may fall short in terms of re-locating the recorded files. Using a real system trace, the experimental results showed that the performance speedup compared with the base case without NAND flash memory is between 2 and 4. It is a conceptually simple technique for dramatically improving disk performance. As a light-weight solution, the predictor can run on the same CPU using their spare CPU power without causing too much CPU and memory overhead. Because frequency distribution is relatively stable, it is unnecessary to keep swapping the files between the flash memory and the disk. Instead, it might be acceptable to run the training once a month to update the predictive model.

7. CONCLUSION

In this paper, we have described KDD technology embedded in a commercial product (An Evolutionary Storage System integrating file frequency predictor, FFP). We showed that in some systems file attributes are strongly related to its access frequency. We have also presented a method of predicting file access frequency. The decision-tree-based predictor FFP is supposed to be a bridge linking the past (attributes at creation) and the future (frequency). During the training period a predictor for file frequency is constructed from observations of file system activity in trace. The file system can then use this model to predict the frequency of newly-created files. The frequency information could be used to decide what files to keep in a flash memory. The trace-driven experimental results indicate that the performance speedup due to the prediction-enabled optimization is 2-4 compared with base case without flash memory.

ACKNOWLEDGEMENT

We thank Jonathan Ledlie and other SOS (SOS project, 2006) project researchers for providing us the simulation traces and the heuristics used in our design.

REFERENCES

T. M. Kroeger and D. D. E. Long, 1999, The Case for Efficient File Access Pattern Modeling, *in Proceedings of the Seventh Workshop on Topics in Operating systems*.

Daniel Ellard, et al., 2003, Attribute-Based Prediction of File Properties, *Harvard Computer Science Group Technical Report TR-14-03*.

Michael Mesnier, et al., 2004, File Classification in Self-* Storage Systems, *Proceedings of the First International Conference on Autonomic Computing (ICAC-04), New York*.

Carl Staelin and H. Garcia-Molina, 1990, Clustering active disk data to improve disk performance, *Tech. Rep. CS-TR-283-90, Dept. of Computer Science, Princeton Univ., Princeton, N.J.*

Pei Cao, et al., 1996, Implementation and Performance of Integrated Application-Controlled File Caching, Prefetching, and Disk Scheduling, *ACM Transactions on Computer Systems, 14(4):311–343*.

Daniel Ellard, et al., 2003, Passive NFS Tracing of Email and Research Workloads, *In Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST'03), pages 203–216, San Francisco, CA*.

Daniel Ellard, et al., 2003, The Utility of File Names, *Technical Report TR-05-03, Harvard University Division of Engineering and Applied Sciences.*

Gregory R. Ganger and M. Frans Kaashoek, 1997, Embedded Inodes and Explicit Grouping: Exploiting Disk Bandwidth for Small Files, *In USENIX Annual Technical Conference, pages 1–17*.

James T. McClave, et al., 1997, *Statistics*, Prentice Hall.

B. Bouchon-Meunier, et al., 1995, Fuzzy logic and soft computing, *Singapore, River Edge, NJ: World Scientific*.

Sape Mullender and Andrew Tanenbaum, 1984, Immediate Files, *In Software – Practice and Experience, number 4*.

Keith Muller and Joseph Pasquale, 1997, A High Performance Multi-Structured File System Design, *In Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP-91), pages 56–67, Asilomar, Pacific Grove, CA.*

R. Hugo Patterson, et al., 1995, Informed Prefetching and Caching, *In ACM SOSP Proceedings*.

Mendel Rosenblum and John K. Ousterhout, 1992, The Design and Implementation of a Log-Structured File System, *ACM Transactions on Computer Systems, 10(1):26–52*.

SOS project home page, 2006, http://www.eecs.harvard.edu/sos/index.html.

IBM Company home page, 2006, *IBM Mainframe Introduction*,

http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_intro.html

Frank Wang, 2006, et al., Speeding Up a Magnetic Disk by Clustering Frequent Data, *IEEE Transactions on Magnetics, eg13*.

Daniel Ellard and Margo Seltzer. 2003, New NFSTracing Tools and Techniques for System Analysis, *In Proceedings of the Seventeenth Annual Large Installation System Administration Conference (LISA'03), pages 73–85, San Diego, CA*.