Proceedings of the International Performance Computing and Communications Conference (IPCCC 2003), Phoenix, AZ, USA: IEEE, April 2003, pp.417-424.

Visualizing Cache Effects on I/O Workload Predictability

Ahmed Amer[†]
Department of Computer Science
University of Pittsburgh
amer@cs.pitt.edu

Alison Luo[‡] Newton Der[‡] Darrell D. E. Long[†] Alex Pang[‡]
Department of Computer Science
University of California, Santa Cruz
{*alison,darrell,newtond,pang*}*@cs.ucsc.edu*

Abstract

We describe our experience graphically visualizing data access behavior, with a specific emphasis on visualizing the predictability of such accesses and the consistency of these observations at the block level. Such workloads are more frequently encountered after filtering through intervening cache levels and in this paper we demonstrate how such filtered workloads pose a problem for traditional caching schemes. We demonstrate how prior results are consistent across both file and disk access workloads. We also demonstrate how an aggregating cache based on predictive grouping can overcome such filtering effects. Our visualization tool provides an illustration of how file workloads remain predictable in the presence of intervening caches, explaining how the aggregating cache can remain effective under what would normally be considered adverse conditions. We further demonstrate how the same predictability remains true with physical block workloads.

1 Introduction

Storage systems and I/O behavior are a major performance bottleneck, and despite extensive study, I/O workloads are typically difficult to synthetically reproduce with any degree of accuracy [6]. To overcome the performance bottlenecks of storage systems, predictive techniques have been applied, attempting to infer future data accesses from prior requests. The viability of such approaches depends on the inherent predictability of the access pattern. In this paper we describe a mechanism for analyzing and visualizing the predictability of workloads, with an emphasis on the effects of intervening caches. When an observed workload is the result of cache misses then we are presented with a modified workload that has been filtered through a prior, intervening, cache. This is an increasingly common scenario, particularly with web proxies, browser caches, and more complex storage systems. Prior work has demonstrated that such scenarios can quickly render traditional caching algorithms useless [1]. In that work we further demonstrated that a promising caching mechanism based on predictive grouping, the *aggregating cache* [2, 3], was able to maintain good performance in spite of such cache filtering. Utilizing our system for visualizing I/O predictability [12–14], we are now able to illustrate how predictive caching policies can succeed in spite of intervening cache filtering, thanks to the absence of any decrease in predictability. Futhermore, we demonstrate that these observations are equally valid for physical disk blocks as for file requests.

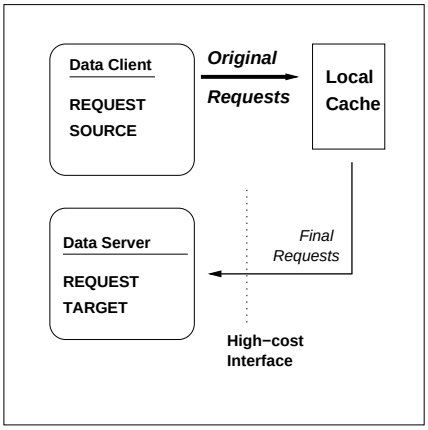
[†]Supported in part by the National Science Foundation award CCR-9972212, and by the USENIX Association.

[‡]Supported in part by the National Science Foundation award ACI-9908881, NASA award NCC2-1260, and LLNL Agreement No. B347879 under DOE Contract No. W-7405-ENG-48.

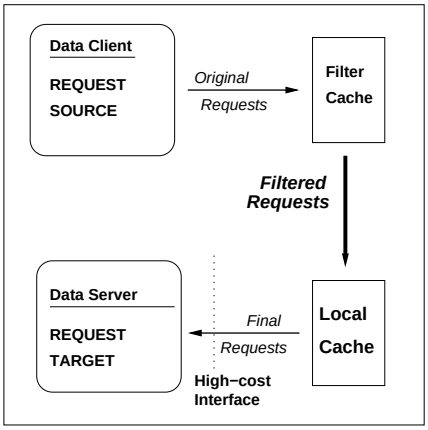
2 Cache Filtering

Caching algorithms are most often tested as a single layer between a source of requests and a target. It is most common for a cache to be used as an intermediate store, one that is faster than a larger, slower, main store. We refer to this as the classical single-stage model (see Figure 1(a)), where the cache is tested against a workload directly. And yet, with the growing complexity and scale of distributed data storage, it is increasingly unlikely that an observed workload will remain unchanged from its original source. More often we will observe workloads that are the result of misses from caches that lie between our local cache and the original source of the data requests. We refer to such intervening caches as *filter* caches that modify the workload (see Figure 1(b) for a simple two-stage model).

The filter cache is logically closer to the request source (data client) than our local cache. Our main concern is the effectiveness of such a *cache* (second stage cache) in light of the filtering effects of the *filter* (first stage cache). This scenario, two comparable caches with a limited cost for inter-cache access, can be found in many distributed computing environments. Examples include mobile file hoards when on a LAN, where the access requests to a server cache have already been pre-filtered through an increasingly large intervening cache – the mobile computer's local storage. Another example includes distributed storage systems having large client-side caches, and high-speed network interfaces to a high performance storage server. In



(a) *Simple one-stage caching model.*



(b) *Two stage cache configuration.*

Figure 1: Single and two stage caching models.

such a system, client accesses to local storage are comparable to remote data retrievals, and so we are questioning the usefulness of server-side caching. In a system like Sprite [16], with a high performance network, we would therefore be considering the effects of a client cache on a server cache's hit rate.

When we consider a filter that employs LRU replacement, and which is comparable in size to the cache, we find that traditional caching schemes like LRU and LRU are quickly rendered useless, with miss rates rapidly approaching 100%. Intuitively this is because such schemes depend solely on locality of access to succeed, a property that is increasingly invalidated as a workload is filtered through an LRU caches of increasing sizes. We've found this behavior

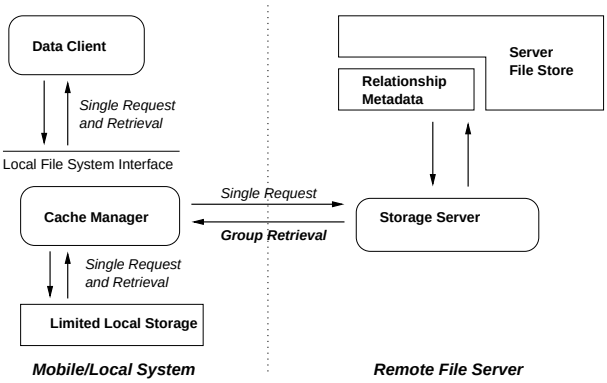


Figure 2: General aggregating cache model.

to be consistent across a variety of trace-based workloads, recorded from real-world storage system behavior. Recent research has explicitly considered this problem [5, 20], attempting to guarantee exclusion between different caches, but at the cost of the complexity and communication overheads of cooperation. A successful alternative we have discovered [1] is to use a predictive cache such as the *aggregating cache* [2, 3], which can successfully exploit workload predictability, which in turn we will show to be unharmed by intervening cache stages.

The Aggregating Cache – The most prominent feature of the aggregating cache is the retrieval of groups of related files from the remote storage server. Although originally intended to reduce the impact of high latency data retrieval requests, the aggregating cache was found to increase cache hit rates in addition to its intended use of reducing the aggregate number of demand fetches [2]. Figure 2 presents a conceptual view of this scheme. File access requests go through the local file system interface and, if not satisfied locally, are forwarded through a local cache manager to the file store managed at a remote server. The major difference from prior art in distributed caches is the mechanism of maintaining server-side relationship information, and its use to retrieve multiple related files per request. This allows the client and server to transparently utilize available bandwidth to fetch groups of files based on observed access patterns.

The server component maintains per-file relationship information, keeping track of a strictly limited group of related files. When a client is forced to perform a high-latency remote request, the server and client components of the aggregating cache can cooperate to opportunistically retrieve a group of related files. We build such groups by tracking a fixed number of successors for all files accessed. A successor is simply a file accessed immediately following the current file, and this information can easily be maintained as file metadata. Because it exploits inter-

file relationships, such a cache remains resilient to the effects of workload filtering, as we have found such filtered workloads to often exhibit more predictable access patterns than the original access sequence.

3 Visualizing Predictability

To visualize the effects of intervening caches we utilized a purpose-built application, *VIP*, and used it to generate *cache-frequency plots*. These plots provide a graphical representation of successor predictability across all objects in workload, while simultaneously indicating their frequency of access. The measure of predictability we use is a conditional self-information metric we refer to as *successor entropy*.

3.1 Successor Entropy

For access predictability we use conditional probability as our metric. $P(s_i|B)$ is simply the probability of occurrence of symbol s_i given the knowledge that we are in state B. The state B is simply taken as the knowledge of the current file or block being accessed.

$$H = -\sum_{i=1}^{m} P(s_i|B) \cdot log(P(s_i|B)) \quad (1)$$

In this work, we examine sets of file and disk access traces. Each trace consists of a sequence of files or blocks that were accessed. For each object f, all recorded successors of f are candidates for a prediction. If file f is used as the condition in Equation 1, then we have $H(f)$ as a measure of the amount of disorder among its immediate successors. The higher this value, the less promising the possibility of an accurate successor prediction. Simply plotting the values of $H(f)$ for each object encountered in a trace would provide a histogram that gives a good idea of the variation in conditional entropy over the file space.

3.2 Cache Frequency Plots

Our visualization system, *VIP* [12–14], can be used to display the behavior a storage system given arbitrary parameter changes. In this paper we will use its abililty to generate *cache frequency plots*. Successor entropy is calculated for each object in the workload. These values are then used to build a histogram for each workload. *Cache frequency plots* are representations of how the successor entropy histogram varies as the workload is changed by filtering through varying intermediate cache sizes. In addition to plotting this data as a 3-D surface, VIP can simultaneously represent a fourth dimension through the use of color. In our cache frequency plots, the frequency of access for each object (file or disk block) is mapped to a color on a perceptual color range. Interactive interrogation of these plots is also possible using VIP.

VIP allows the user to interactively select specific data points for which it will display detailed textual information. The interactive feature of VIP include:

1. **Point identity**: Data points in our system are associated with real files, and by interactive picking of data points we are able to identify the specific name associated with object corresponding to a data point.

2. **Identity equivalence**: Plotting ordered histograms of successor entropy allows us to view a consistent surface, but this also means that fixed points on the *x* axis do not correspond to one fixed file. When a point is selected in VIP, all other points on the surface corresponding to the same object are indicated with red markers (see Figure 7(b) below). This allows the user to confirm if particular files' predictability varies with changes in the variable parameter (the *y* axis, which in this paper is used to represent filtering cache size).

3. **Value equivalence**: While it is useful to indicate points referring to the same object, VIP also allows us to indicate all data points that share a common range of values (see Figure 7(a) below). In our examples these values represent the frequency of access to the corresponding objects, and are indicated by color.

These interactive features were very useful in explaining workload behavior. Using value equivalence we were able to demonstrate that a large proportion of the most frequently accessed objects are in fact the most predictable.

4 Experimental Results

In this section we present our experimental results. We start by describing the different traces that we have used for our experiments. We then go on to describe the adverse effects of cache filtering on second-stage caches, and how the aggregating cache is resilient to these effects. We conclude this section with a presentation of cache frequency plots that demonstrate how disk and file workloads remain predictable in spite of filtering effects, explaining how the aggregating cache remains effective while traditional non-predictive caching schemes fail.

4.1 Workloads

We tested our system with two different sets of file access traces, and one set of physical disk traces. The first set of file accesses were drawn from file system traces gathered using Carnegie Mellon University's DFSTrace system [15]. These traces provided information at the system-call level, and represent the original stream of access requests. These requests were filtered to represent initial accesses to different files, *i.e.*, accesses are based on file open requests. This provides coarse granularity for the analysis, focusing on patterns of file requests, more representative

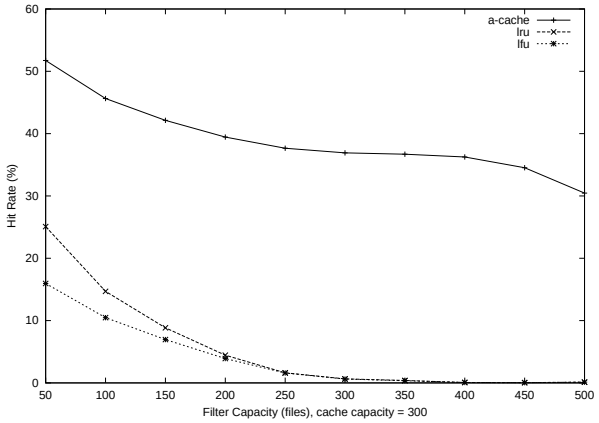
of file hoarding scenarios. For a finer level of analysis, we used a second set of file traces provided by Roselli at the University of California, Berkeley [17]. These traces provided details of individual requests for file data. To eliminate any interleaving issues, these UCB traces were processed to represent the workloads of individual workstations, and simulations were run against both instructional and research machines. Our third set of traces are physical block-level accesses drawn from Hewlett Packard's SRT traces [18]. The traces were processed to represent individual requests for disk blocks, and durations of a day to several weeks were analyzed.

4.2 Cache Filtering Effects

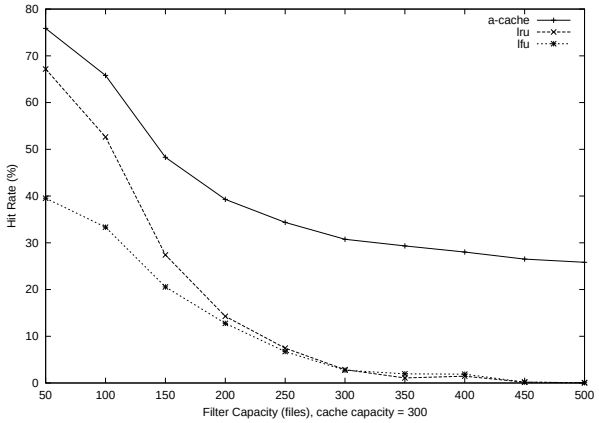
The most interesting results occur when the filter cache capacity grows comparable to the local cache size, with the usefulness of a traditional cache disappearing as the filter capacity exceeds the cache size. For our first set of results we will fix an arbitrary cache size, and observe the effect of increasing the intermediate filter capacity on cache performance. For the following graphs we chose to fix filter capacity at 100 objects.

Figure 3 shows the effects of varying the capacity of the filter on the hit rate of our cache. We compare three cache management schemes: LRU replacement, LFU replacement, and a basic aggregating cache (that tracks and retrieves groups of up to five related files). It is no surprise that LRU outperforms LFU replacement, but the most important observation from the figure is how rapidly the performance of the cache degrades. As the filter size approaches the fixed cache size, we see a dramatic drop in the hit rate for our cache. This is consistent both for the dedicated workstation *workstation*, and the more heavily loaded system *server*. Regardless of the nature of the request source (multi-user or dedicated system) this degradation appears very rapidly, and both LRU and LFU caching quickly become useless. In contrast, the aggregating cache maintains consistent performance, and shows a much milder degradation in hit rate. All independent locality of reference is quickly masked by the intervening cache, rendering straightforward LRU caching useless while the aggregating cache manages to maintain a higher hit rate in spite of this. This is thanks to the ability of this scheme to capture inter-file relationships. Although the intervening cache masked all observable locality for LRU and LFU, these schemes assume an independence of access. Fortunately, the interdependence among file access events is not completely masked by filtering, allowing the aggregating cache to sustain a hit rate even when requests are filtered through a filter larger than the cache.

It should be noted that the aggregating cache is being supplied with exactly the same information as the LRU and LFU schemes, *i.e.*, there is no cooperation between the



(a) *workstation*

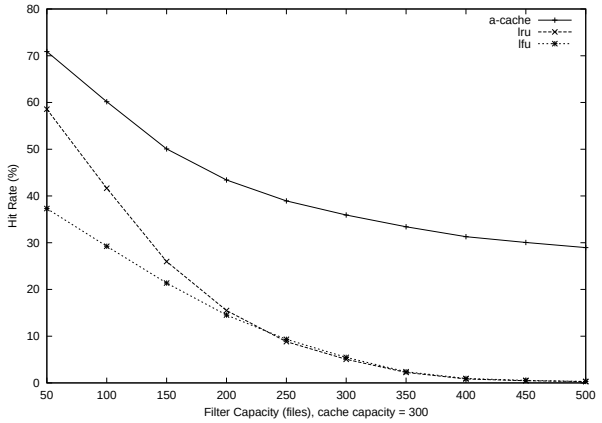


(b) *server*

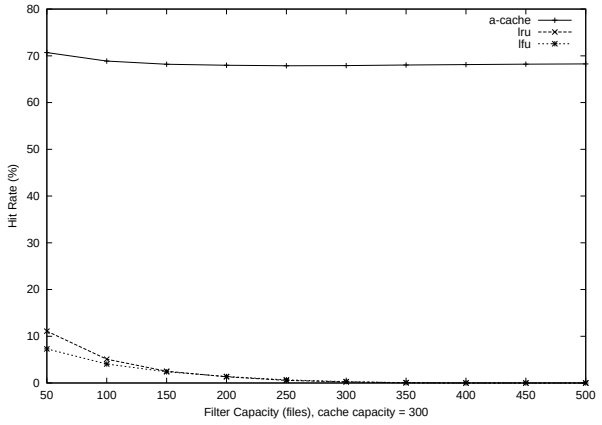
Figure 3: CMU trace cache hit rates for varying filter sizes.

cache stages. Although the design of the aggregating cache allows forwarding of relationship information, gathered at the data client (request source), to the storage server, we do not use this feature in these experiments. If this scheme were used we would expect higher hit rates (above 90% for these workloads), equivalent to an aggregating cache free of the filtering effects of an intervening cache. These results represent the performance of an aggregating cache when confronted with the same problematic access behavior as presented to the LRU and LFU schemes.

A trend observable in Figure 3 is the rate of decline for the LRU/LFU caches *vs.* the aggregating cache. Other experiments have shown us that the *workstation* workload tends to be more predictable than *server*. The *workstation* trace, with fewer users and subsequently fewer inde-



(a) *Instructional Worsktation*



(b) *Research Workstation*

Figure 4: UCB trace cache hit rates for varying filter sizes.

pendent sources of requests, degrades more rapidly due to filtering, and yet the aggregating cache is better able to maintain hit rate. While with *server* we see a more gradual degradation in LRU and LFU, while the aggregating cache is slightly less adept at tolerating the filtering. This supports the reasoning that the aggregating cache's performance is due to its ability to capture higher-level dependencies between file access events. More independent accesses result in reduced filtering impact and reduced improvement through use of the aggregating cache.

The CMU traces are most useful for their extensiveness, as they cover well over a year of recorded workloads, but may be too old for to be fully representative of current workloads. Also, they often lack recordings of more detailed access events, *e.g.* specific read and write system calls. For these reasons we repeated our experiments on the UCB traces [17]. Specifically we present results of a typical research workstation and an instructional system. Figure 4 illustrate the same results as Figure 3 but using the UCB traces.

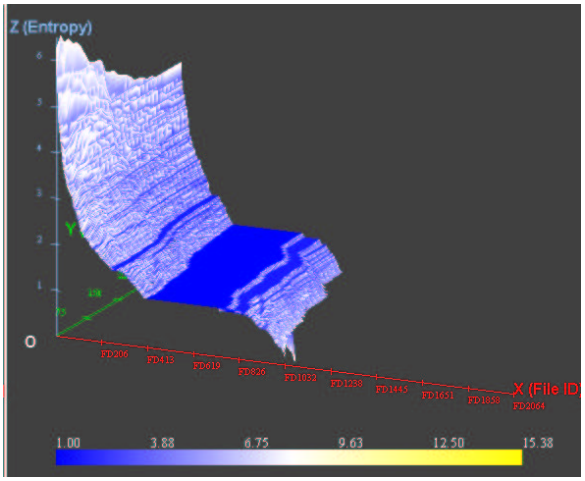
Both figures demonstrate the same trends we observed for the CMU traces, with a notably higher performance for the aggregating cache, especially for the more predictable research workload. In fact, the aggregating cache could be considered to have been only slightly affected by the growing size of the filter, while the LRU and LFU are even more seriously affected. Our hypotheses appear to hold more strongly for the more recently recorded workloads. To understand the reasons for this behavior we look at the results of our visualization system in the following subsection.

4.3 Visualizing Cache Filtering

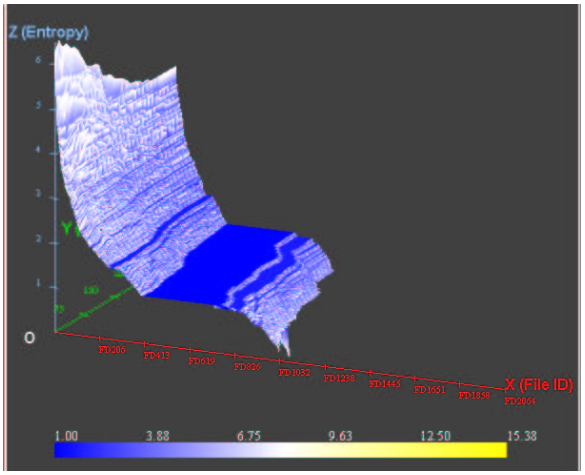
For caching effects, we used *VIP* to visualize the effects of increasing cache sizes on the workload. This was done by filtering the workloads through a simulated cache of capacity ranging from 0 to 300 onjects, and then plotting the conditional file entropy histograms for each cache capacity, using color to indicate the frequency of access for the corresponding file. A capacity of 0 is equivalent to the original access sequence, while a cache of size 300 represents a large enough cache capacity to render simple independent probability-based predictions useless.

In Figure 5 we present the results of these experiments for file access sequences of approximately a month. The first observation from Figure 5 is how predictable the file accesses remain in spite of increasing the capacity of intervening caches. This is contrary to commonly held views of file access behavior, where caches are expected to simply result in a greater degree of unpredictability. Noting the distribution of frequently accessed files – they occur at ranges of very high predictability – we can begin to see how a a succession-based predictive caching scheme was able to maintain good performance in spite of the intervening caches. The following observations hold for the file access traces in Figure 5: First, many high-frequency files have the most predictable successor behavior; Secondly, when no or little cache filtering occurs (at smaller capacities) there are a larger proportion of frequently accessed yet unpredictable files. The latter point clarifies why a predictive cache remains effective in spite of intervening caches. Intervening caches act as a consistent filter that masks most localised accesses. What is observed is in fact the transitions among working sets, inherently less likely to be affected by transient events and more likely to represent true access behavior.

These observations are also true when considering disk block access patterns. Figure 6(a) shows a day-long trace of disk access behavior, while Figure 6(b) shows the same

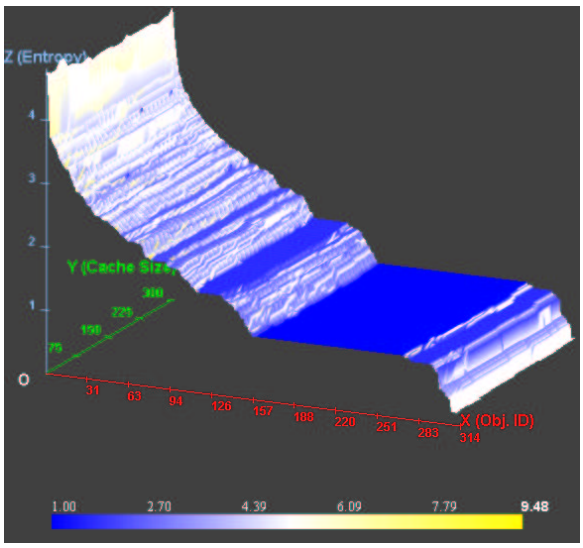


(a) *workstation*

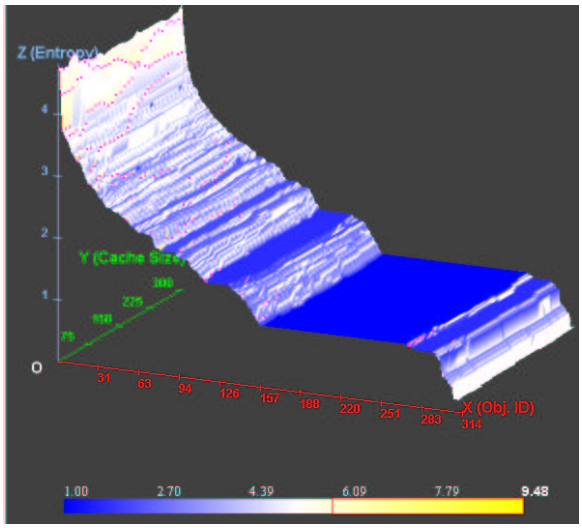


(b) *server*

Figure 5: Cache-frequency plots of month long CMU file access traces. All points are colored by file access frequency.



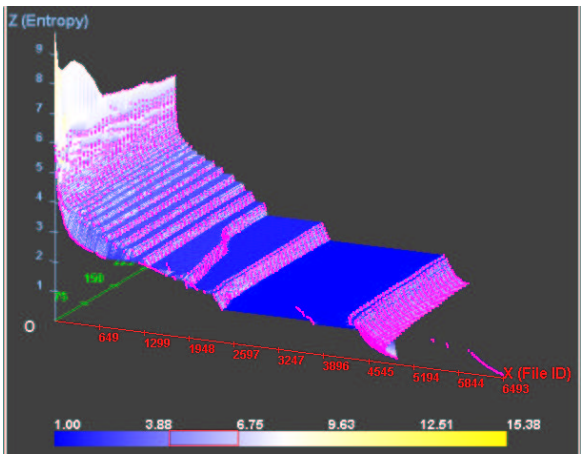
(a) day



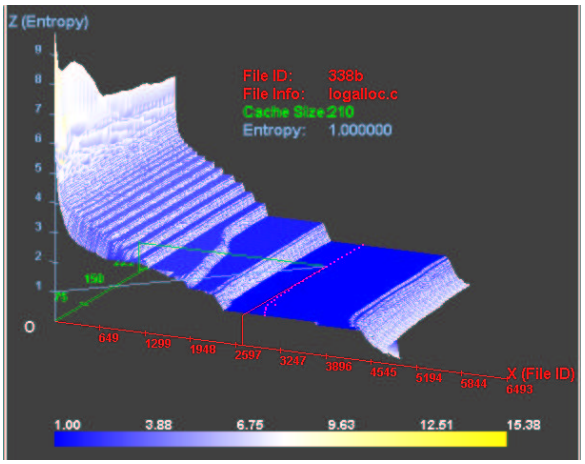
(b) week

Figure 6: Cache-frequency plots of SRT disk access traces. All points are colored by block access frequency.

view for disk accesses over an entire week. As with file access patterns we can see that most popular files are widely distributed across the range of predictable files. Specifically, aside from the original workloads (with cache size 0), there is limited correlation between predictability and access frequency (we see lighter points across the range of files and predictability values). This is especially important as it clearly demonstrates that many of the most popular disk blocks are also very predictable in their associated access behavior. This observation can be further supported when the interactive features of VIP are employed. Figure 6(b) shows the selection of all blocks with a relatively high associated access frequency. The selection is made using a highlight-bar across the frequency legend. All points that correspond to blocks with a popularity falling in that range are indicated in red (darker single points in the hardcopy figure). Figure 7(a) shows similar selection for the file access workloads. The selection is for



(a) *Selecting a frequency range.*



(b) *Selecting a single data point.*

Figure 7: Interactive features. Selecting a single point provides associated file/object information, while selecting a frequency range will highlight all point on the surface that fall in that range.

less frequently accessed files, and therefore covers many more points (again distributed across the range of files). This is consistent with the expected degree of skew in file access frequencies.

A further insight gained from the interactive features of VIP was a direct result of identifying the specific files contributing to data points in the file access traces. A feature of these cache-frequency plot was the relatively large flat regions, particularly in the workstation workload (Figure 5(a)). Although it was clear these were probably a large number of files sharing a common predictability, it was not clear why this would be the case. Selecting points in one such region showed them to be files supporting code development; libraries, code source files, and headers. This explains the existence of large groups of files exhibiting near-identical behavior, which in turn implies neas-identical predictability. All such files belonged to one or more development projects, where accesses would be driven by a common makefile, resulting in highly deterministic and common access behavior. The selection of one such file (and visual confirmation that all other data points from that file fall in the same range) is presented in Figure 7(b).

5 Related Work

Our work on the aggregating cache has drawn from work in distributed file systems, and predictive prefetching, but our consideration of cache filtering effects is novel in this area. Griffioen and Appleton presented a file prefetching scheme based on graph-based relationships [7]. The first proposed application of data compression techniques to file access prediction was presented by Vitter and Krishnan [19]. Later work by Kroeger and Long [9] compared the predictive performance of different algorithms, including Griffioen and Appleton's scheme, and more effective schemes based on context modeling and data compression [8, 10]. Yet another technique, using program-based file prediction based on pattern matching, was also presented by Lei and Duchamp [11]. All these studies focused on the prediction of file accesses using models that could observe requests at the source, client-side models.

Although prior work has considered the effects of multiple cache levels [5, 20, 21], we believe our work to be the first to provide a means to directly illustrate the effects of caching on the predictability of disk or file access patterns. Our results explain why a predictive cache can remain effective without communicating with prior cache levels, but prior work has attempted to guarantee cache exclusion without a communication overhead and without predictive policies [4]. This work by Ari *et al.* depended on the use of adaptive caching policies to dynamically optimize the behavior of multiple non-communicating caches.

6 Conclusion

Caches are useful for reducing requests to slower data storage devices, but without cooperation intervening caches have traditionally been considered detrimental to the operation of caches at later stages. We have demonstrated that this assumption is clearly true for traditional caching schemes that only exploit locality, but predictive caches remain effective. Using *VIP* to generate cache-frequency plots, and its interactive features, we were able to draw two valuable insights into the nature of file and

disk workloads: frequency is not correlated with unpredictability, and caching does not degrade the predictability of the access patterns. The first point contradicts the assumption that frequently accessed objects are less likely to have easily predictable successors. The second explains why predictive caches can work without the cooperation of intervening caches – intervening caches may mask locality, but do not render the access patterns less predictable. On the contrary, they would appear to filter out transients, leaving only the more predictable working-set transitions.

Acknowledgments

We are especially grateful to Thomas Kroeger and Randal Burns for valuable feedback and discussions. We are grateful to all the members of the Storage Systems Center, for their continuous feedback, support and valuable discussions. Our lengthiest 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 providing the UCB traces used in this study, and John Wilkes of HP for providing us with the SRT traces.

References

[1] A. Amer and D. D. E. Long, "Adverse filtering effects and the resilience of aggregating caches," in *Proceedings of the Workshop on Caching, Coherence and Consistency (WC3 '01)*, (Sorrento, Italy), ACM, June 2001.

[2] 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.

[3] A. Amer, D. D. E. Long, and R. C. Burns, "Group-based management of distributed file caches," in *Proceedings of 22nd International Conference on Distributed Computing Systems (ICDCS 2002)*, (Vienna, Austria), IEEE, 2002.

[4] I. Ari, A. Amer, E. L. Miller, S. Brandt, and D. D. E. Long, "ACME: Autonomous Caching using Multiple Experts," in *Workshop on Distributed Data and Structures (WDAS 2002)*, (Paris, France), p. (to appear), Carleton Scientific, Mar. 2002.

[5] M. Busari and C. Williamson, "Simulation evaluation of a heterogeneous web proxy caching hierarchy," in *Proceedings of the 9th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '01)*, (Cincinnati, OH), pp. 379–388, IEEE, Aug. 2001.

[6] G. R. Ganger, "Generating representative synthetic workloads: An unsolved problem," in *Proceedings of the Computer Measurement Group (CMG) Conference*, pp. 1263–1269, Dec. 1995.

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

[8] T. M. Kroeger and D. D. E. Long, "Predicting future file-system actions from prior events," in *Proceedings of the 1996 Usenix Winter Technical Conference*, (San Diego), pp. 319–328, Jan. 1996.

[9] 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.

[10] 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.

[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] A. Luo, A. Amer, N. Der, D. D. E. Long, and A. Pang, "Visualizing file system predictability," in *Works In Progress at IEEE Visualization 2001*, (San Diego, CA), Oct. 2001.

[13] A. Luo, A. Amer, N. Der, D. D. E. Long, and A. Pang, "Visualizing the predictability of file access patterns," in *Proceedings of IASTED Computer Graphics and Imaging (CGIM)*, (Kauai, Hawaii), Aug. 2002.

[14] A. Luo, A. Amer, S. Speidel, D. D. E. Long, and A. Pang, "Visualizing I/O predictability," in *Proceedings of the First Symposium on 3D Data Processing Visualization Transmission*, (Padova, Italy), June 2002.

[15] 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.

[16] M. N. Nelson, B. B. Welch, and J. K. Ousterhout, "Caching in the Sprite network file system," *ACM Transactions on Computer Systems*, vol. 6, no. 1, pp. 134–154, 1988.

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

[18] C. Ruemmler and J. Wilkes, "UNIX disk access patterns," in *Proceedings of the Usenix Technical Conference*, (San Diego, CA), pp. 405–420, Usenix Association, Winter 1993.

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

[20] T. M. Wong, G. R. Ganger, and J. Wilkes, "My cache or yours? Making storage more exclusive," tech. rep., CMU-CS-00-157 Carnegie Mellon University, Nov. 2000.

[21] Y. Zhou, J. F. Philbin, and K. Li, "The multi-queue replacement algorithm for second level buffer caches," in *Proceedings of the 2001 USENIX Annual Technical Conference*, (Boston, MA), June 2001.