STEP: Self-Tuning Energy-Safe Predictors *

James Larkby-Lahet Ganesh Santhanakrishnan Ahmed Amer Panos K. Chrysanthis
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260 U.S.A.
{james, ganesh, amer, panos}@cs.pitt.edu

ABSTRACT

Data access prediction has been proposed as a mechanism to overcome latency lag, and more recently as a means of conserving energy in mobile systems. We present a fully adaptive predictor, that can optimize itself for any arbitrary workload, while simultaneously offering simple adjustment of goals between energy conservation and latency reduction. Our algorithm, STEP, achieves power savings on mobile computers by eliminating more data fetches, which would otherwise have caused excess energy to be consumed in accessing local storage devices or using the wireless interface to fetch remote data. We have demonstrated our algorithm to perform as well as some of the best access predictors, while incurring almost none of the associated increase in I/O workloads typical of their use. Our algorithm reduced average response times by approximately 50% compared to an LRU cache, while requiring less than half the I/O operations that traditional predictors would require to achieve the same performance, thereby incurring no energy penalty.

Categories and Subject Descriptors

D.4.3 [**Operating Systems**]: File Systems Management—*Distributed File Systems*; H.2.4 [**Database Management**]: Systems—*Distributed databases, Query Processing*; H.3.m [**Information Storage and Retrieval**]: Miscellaneous—*Caching, Prefetching*

General Terms

Algorithms, Design, Simulation, Performance

Keywords

Mobile Computing, Power Management, Adaptive Caching, Prediction, Prefetching

* This work has been supported in part by the National Science Foundation ITR Medium Award ANI-0325353 and a Pitt-Startup 2002 Research and Development Grant from the University of Pittsburgh to the third author. The first author has been funded by a Research Experiences for Undergraduates (REU) program of the University of Pittsburgh.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MDM'05, May 09-13, 2005, Ayia Napa, Cyprus
(c) 2005 ACM 1-59593-041-8/05/05....$5.00

1. INTRODUCTION

The perceived lag in the performance of data storage technologies compared to processor performance has resulted in numerous efforts to overcome access latencies through predictive or prefetching caches. Access latencies to remote data are incredibly difficult to reduce, and are often restricted by physical limitations (*e.g.*, the speed of light and interventing distance), as such I/O access latencies tend to be one of the most insurmountable performance obstacles in modern computing. While insurmountable, this problem is often avoided through the use of appropriate caching schemes. In mobile environments the performance considerations go beyond simple speedups and service time reductions, but also include avoiding excessive energy consumption.

Caches generally attempt to guarantee that most data requests are for data that is being held in main memory or local storage, negating the need to perform I/O, or a remote data retrieval. Prefetching caches on the other hand attempt to achieve greater perceived performance through the prediction of future data requests, and the retrieval of data items before they are requested. The success of such schemes depends on more than the accuracy of the predictive algorithm applied, but also on the timeliness of the prediction (was it made early enough to be of use), and the impact of incorrect predictions (if they result in unnecessary I/O operations). A successful prefetcher has the potential to improve performance and almost completely mask data retrieval delays, while an unsuccessful predictor can seriously degrade a storage system's performance by creating excessive I/O load for prefetching predicted data items that are never used. Thus, for mobile applications any potential failure has an added disadvantage in that excessive cache misses will either result in more frequent data requests to local storage, or through the network interface, both of which would results in unnecessary energy consumption. Finally a cache's performance, whether it be a traditional or prefetching cache, is dependent on how well it's particular caching heuristics react to the workload at hand.

In this paper we examine the question of whether predictive prefetching is desirable in the context of mobile environments such as a mobile environment. Our contribution is a novel algorithm, **S**elf-**T**uning **E**nergy-safe **P**rediction (*STEP*), which indeed shows that caching with prefetching can be beneficial in energy constrained evironments. Prediction has been proposed as a means to improve storage performance by prefetching data and reducing response times, or by predicting disk activity and appropriately spinning down disk devices, or more recently by prefetching data to extend the duration of device idle periods [15, 30, 18, 9].

STEP is a self-tuning data access predictor, that not only offers good predictions of future accesses, but does so in a fully adaptive manner that can react to arbitrary workloads. STEP achieves this goal while automatically guarding against the over-zealous prefetching of unnecessary data, avoiding any potential energy overheads. In order to evaluate our predictor we have built an experimental testbed in which we have used a set of real traces as test workloads. We demonstrate the success of STEP using this set of widely varying, block-level, data access traces.

The rest of the paper is organized as follows. In the next section, we describe our Self-Tuning Energy-safe Predictors (STEP) policy. We go on to present our experimental results in Section 3. In Section 4, we discuss related work and conclude with a brief summary and directions for future work in Section 5.

2. SELF-TUNING PREDICTION: STEP

STEP is an algorithm that predicts likely future data requests based on observing prior requests. Basically, it identifies the current data access and produces a set of likely subsequent requests. The self-tuning nature of STEP derives from the fact that it continuously evaluates the accuracy of its own likelihood estimates, and adjusts its prediction policy accordingly. These likelihood estimates are produced based on a weighted combination of simple prediction heuristics (*prediction criteria*), and STEP's self-adjustment is based on dynamically adjusting the weighting of these heuristics. The likelihoods STEP calculates in this manner are actually for individual data items appearing in immediate succession, *i.e.*, *successor likelihoods*. To use STEP in a prefetching cache, it would be necessary to retrieve an arbitrary number of multiple successors for any given data access. This is possible by expanding a set of immediate successors, into a larger set of *transitive successors*. The main components of the STEP algorithm can therefore be described as: the master algorithm, the specific heuristics (prediction criteria), and the mechanism to fetch multiple successors.

2.1 The Master Algorithm

STEP adjusts its prediction mechanism using a master algorithm and weight-update scheme based on multi-expert machine learning algorithms [27, 12]. The "experts" are prediction heuristics, based on simple criteria, each of which offers a likelihood estimate for future events.

The experts make predictions, *i.e.*, offer likelihood values for the successors that may be prefetched. The weights of experts represent the quality of their likelihood estimates. Initially the weights are distributed uniformly among all the experts, but as new requests are observed the weights of the experts are updated. The master algorithm proceeds with a weighted average of the experts' individual predictions. We have shown significant performance improvements utilizing this strategy in caching web documents in our earlier works [24, 23]. Since the weights are updated to reflect the quality of an individual experts' likelihood estimates, STEP improves itself via feedback. Figure 1 offers a logical overview of STEP.

2.2 The Experts: Prediction Criteria

Predicting whether a particular item will be requested in the future can be based on many basic factors, such as how often this item was requested in the past, or how recently. Such indicators of future access likelihood are examples of what we refer to as prediction criteria, and constitute the prediction experts we use as components of the master algorithm in STEP.

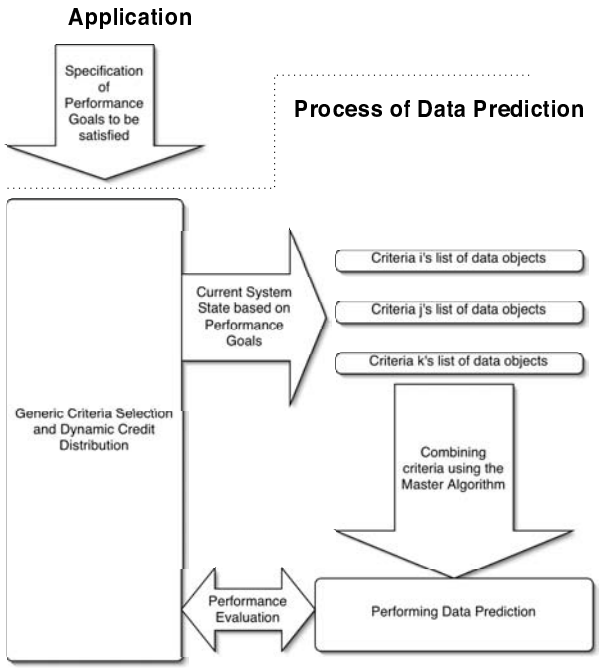


Figure 1: Overview of STEP

The experts that we have utilized in our experiments are simply those of *Frequency*, *Recency* and the *Null Predictor*. Frequency is defined as the number of times a pair of events have occurred in succession divided by the total number of times the first event occurred. For data accesses, this is simply a count-based estimate of how likely the second item is to be observed as the successor of the first.

Recency is a similar heuristic, where succession is estimated based on the temporal proximity of events. Specifically, we define Recency as the complement of the time since an event last occurred divided by the sum of the times since all other candidate events last occurred. The events being considered are the successors of a given item. This gives a quantitative likelihood estimate that is proportional to recency, and conveniently sums to one across the set of potential successors.

The Null Predictor is defined as the prediction that no prediction should be made [4]. Specifically, it says that the next event is likely to be one that has never been observed before as a successor of the given event, and subsequently any attempt at making a prediction would be unnecessary overhead. In this manner, STEP automatically restrains itself from attempting predictions when the workload tends to be unpredictable.

We believe that tracking these criteria is sufficient for data access patterns, since they cover the most generic set of criteria for performing data access prediction. They are further convenient in that they provide a set of properties that are observable for all forms of data access, and make no assumptions regarding the nature or level of observation. Evaluating recency, frequency, and the null predictor require no more of the observed workload than that it take the form of a sequence of requests for identifiable data items. This is in contrast to prediction heuristics that, *e.g.*, assume file-level observation, or that assume a particular data format. Examples of such

heuristics could include the assumption that files sharing a common directory are likely to be accessed together, or that HTML files are likely to be followed by requests for files to which they link. For every observed event, each of our experts effectively calculates its own likelihood estimate for various possible successor events.

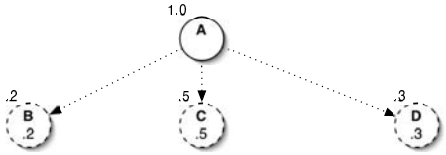
2.3 Predicting Multiple Successors

Prefetching is helpful only if the predicted successor is requested before the actual request. This issue can be overcome by predicting and prefetching several requests into the future [16]. Since each data block has a set of observed successors, the overall data relationships can be represented as a graph, with each directed edge weighted by a likelihood ranging from zero to one. There are two obvious approaches for identifying transitive successors (*i.e.*, successors of successors), One would be a breadth-first approach, which grabs all the successors to a given item before moving on. This gives high assurance that we will have prefetched the next request, at the expense of quite a bit of wasted effort. Alternatively one might use a depth-first approach, which fetches a long chain of sequential events. If the prediction is correct, this is every bit as good as breadth-first, without the excess fetches. However, if a single successor is mispredicted, it is likely that the rest of the chain will be unused as well.

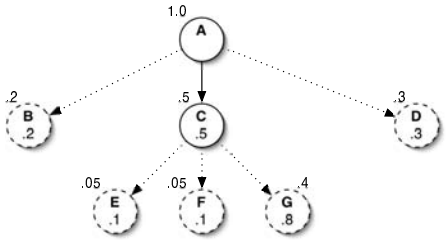
Balanced Expansion provides an alternative that combines these two basic options in a manner that optimizes the likelihood of selected successors being observed as transitive successors [3]. The last observed request serves as the the root of the tree. All of the successors to the request are put into the successor pool. The successor with the highest likelihood is removed from the pool and selected for prefetching. Its successors are then added to the successor pool, with their respective likelihoods multiplied by the prefetched item's likelihood. Again the item with the highest likelihood is removed and prefetched. This continues until we have reached the *search limit*, or if items fall below the *confidence threshold*.

The search limit is the maximum number of items to be prefetched, and is often defined by the workload (demand fetches take priority over prefetch requests), or the problem (*i.e.*, how long the prefetching is allowed to continue). For our experiments we tested search limits of 3, 5 and 10. The confidence threshold is a parameter of STEP, and represents the minimum likelihood required for an item to be prefetched. We used .1, .5, and .9 for our experiments.

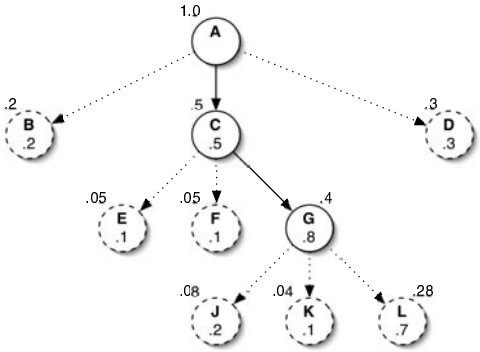
Figures 2(a)-2(d) provide an example of the Balanced Expansion at work. A is the observed item. Its successors B, C, and D are assigned likelihoods by the predictor, as shown in Figure 2(a). The number inside the node represents the likelihood relative to the parent node. The number above the node is is its likelihood relative to the root of the tree, since the root of the tree is an observed event we will simply call this the absolute likelihood. Figure 2(b) shows the tree after C is prefetched and then expanded, as it has the highest absolute likelihood. Its successors E, F, and G are assigned likelihoods by the predictor, and these are also multiplied by C's likelihood to determine the absolute likelihoods. G now has the highest likelihood and is prefetched and expanded, as shown by Figure 2(c). Figure 2(d) illustrates the next round; D has the highest likelihood, so it is prefetched and the tree is expanded to include its successors.



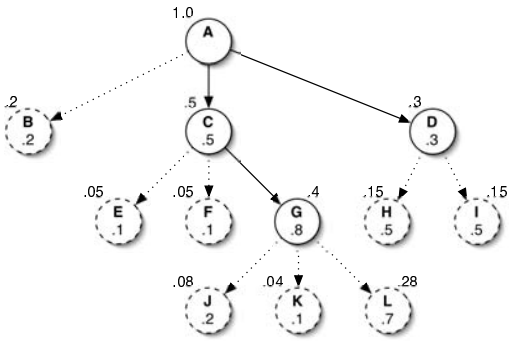
(a) Initial Tree



(b) first expansion, after selecting C



(c) second expansion, after selecting G



(d) third expansion, after selecting D

Figure 2: Balanced Expansion, predicting multiple successors

Experimental Parameters	*Definition*	*Range Tested*
Search Limit	Maximum number of items to prefetch	3, 5, 10
Confidence Threshold	Minimum likelihood required to prefetch	0.1, 0.5, 0.9
Spindown Timeout (s)	Duration of disk idle period before being spun down	1, 5, 10, 15, 20, 25, 30, 45, 60, 90
Spinup Penalty (s)	Delay between disk spun up and service of a request	1, 3, 5
Cache Size (MB)	Size of the LRU cache	1, 2, 4, 8, 16, 32, 64

Table 1: Experimental Parameters

3. EXPERIMENTAL RESULTS

We now describe the workloads used in our experiments, the assumptions we made, and the set of algorithms tested. Our goal is to demonstrate that STEP can realize the performance gains promised by prefetching algorithms, while avoiding the adverse effects that result from the potential for prefetchers to increase I/O load.

3.1 Testbed and Workload Description

We explored the concepts behind our STEP algorithm using block-level disk access traces [22]. The traces we selected represent two very different workloads, one represents that of a single-user workstation (*hplajw*), while the other represents a busy server (*cello*). The workstation traces contain a total of 252,000 accesses that were recorded over a period of two months, while for the server trace we used three days of observed activity, that included 977,000 accesses. We limit our simulations to the read requests so as to focus solely on the benefits of prefetch read optimizations. While these are 1992 traces, the variation in the volume of data, and rates of access, combined with the fact that these are block-level traces motivated our selection of simulated cache sizes. The ranges are representative of possible controller-based, or device-level caches. We selected such cache sizes as small as 1MB, and going up to 64MB. Sizes beyond this tended to exceed the maximum working set size of these particular traces, and thus served as a good upper limit.

We implemented our own cache simulator, and simulated both the behavior of the cache, and its interaction with an underlying disk. In addition to varying the cache size, we also tested for different disk-related parameters. It is especially important in mobile storage that we consider issues such as the energy savings obtained by spinning down a disk to a low-power state, and the time penalty incurred when a spun-down disk needs to be activated and spun up before a request for on-disk data can be satisfied. For this reason, we paid particular attention to timing delays and the power model of the disk. By modifying the power and performance-related parameters of the underlying disk model, we were able to verify our results across a range of possible mobile storage devices. A summary of our experimental parameters is shown in Table 1.

We simulated a total of five alternative cache management algorithms. The first was a *simple LRU* cache without any prefetching, which primarily served as a control. The other algorithms represented four different successor predictors used to prefetch multiple successors. For each of these, the next read request is only processed after prefetching is complete. The second algorithm, the *Last Successor* predictor [2], tracks the last read to follow each block. On subsequent reads, the block is read and then its successor is prefetched. Transitives, that is successors of successors, are also prefetched to provide the multiple successors. This is equivalent to fetching a sequence of most likely successors based solely on the most recent observed successor. The third algorithm is a *Frequency-based Depth First* expansion [1]. It tracks multiple successors for each block, and their frequency of occurrence (observation). A Depth First tree expansion is used to identify multiple successors, with greater priority given to those successors that were observed most frequently. The fourth algorithm *Frequency-based Balanced Expansion* [3], uses an optimized balance of breadth and depth-first expansion of likely successors to identify a group of likely successors. The final algorithm simulated is our STEP policy, which used multiple criteria to evaluate likely successors, and dynamically self-tunes its prediction mechanism.

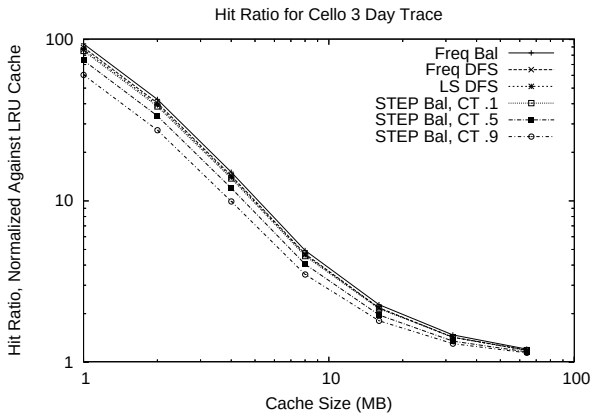
In Figure 3 through Figure 9 we show the performance of prefetchers that used Last Successor (LS DFS), Frequency-based Expansion (Freq DFS), Balanced expansion (Freq Bal), and three different settings of STEP. Each version of STEP presented used a different confidence threshold (CT) for making predictions. This was the minimum estimated likelihood below which no prediction is attempted. For STEP Bal CT .1, the confidence threshold is set to a likelihood of 0.1, implying that STEP will offer predictions even when the estimated likelihood of the prediction being correct is as low as 10%. The higher the confidence threshold, the more conservative the predictions offered by STEP. While the confidence threshold is an adjustable parameter, the specific mechanism STEP uses to produce a likelihood estimate will constantly vary as the workload changes, so this parameter is a more realistic threshold than an arbitrary percentage applied to a fixed algorithm.

3.2 Prefetcher Potential *vs.* Overhead

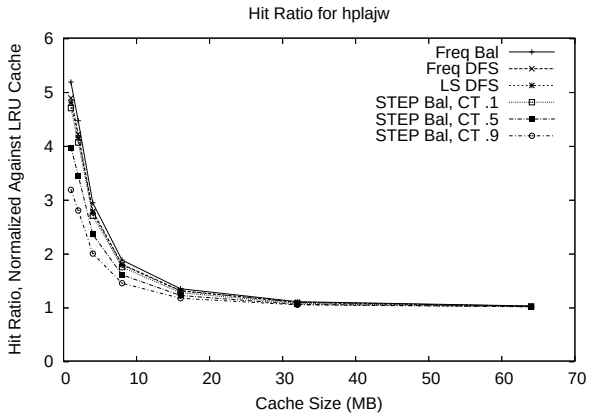
While hit ratios have been commonly used as metrics of cache performance, they neglect to consider the potential increase in workloads created by the prefetching of data. Disk Read-Ratios offer an indication of how much unnecessary I/O is created by a prefetcher. Figure 3 compares the Hit-Ratios across various cache sizes, while Figure 5 shows the Disk Read-Ratios. In both these figures we normalize the results of the prefetching caches against that of the basic LRU policy, giving us an indication of how much relative benefit (or loss) is attributed to adding a particular prefetching algorithm to basic LRU caching. We show the actual range of values for the LRU cache in the caption of each subfigure.

As we can see from Figure 3, when we consider only hit ratios, prefetchers provide dramatic improvements over basic LRU replacement in terms of hit ratio, which taper off as the cache capacity grows large enough to hold most requests. The important thing to note is that not all prefetches are necessarily useful, and while the cache may be exhibiting a higher hit rate, these figures neglect the cost associated with achieving these higher hit rates. If a prefetcher creates more I/O operations than it saves through such higher hit rates, then the performance of the system will likely degrade.

From Figure 3 we see that STEP offers hit rate improvements similar to the competing prefetchers, and appears generally insensitive to the particular confidence threshold setting used. But more im-



(a) cello



(b) hplajw

Figure 3: Hit-Ratio Comparison

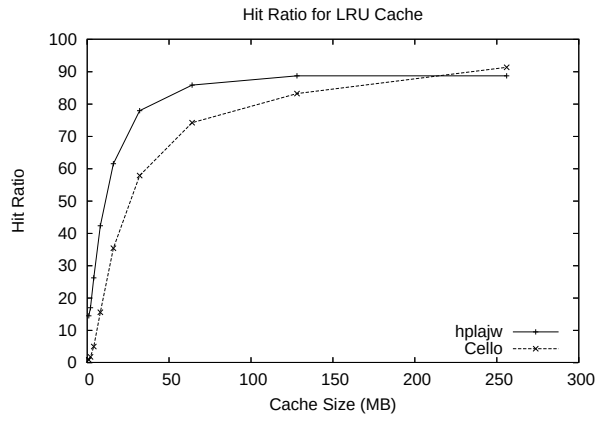
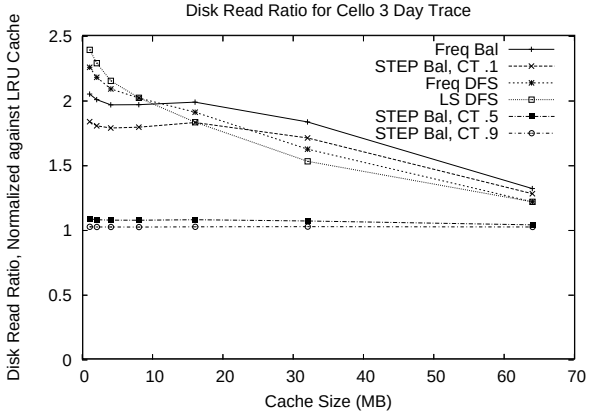
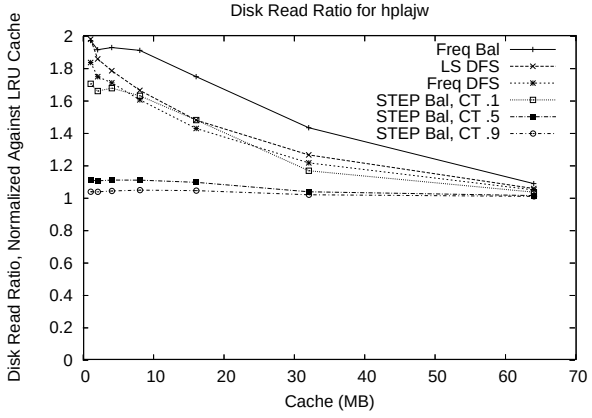


Figure 4: LRU Hit-Ratio (the baseline for Figure 3)



(a) cello



(b) hplajw

Figure 5: Disk Read Ratio Comparison

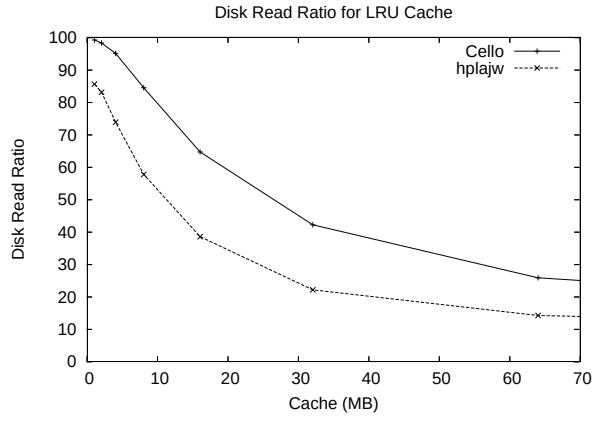


Figure 6: LRU Disk Read Ratio (the baseline for Figure 5)

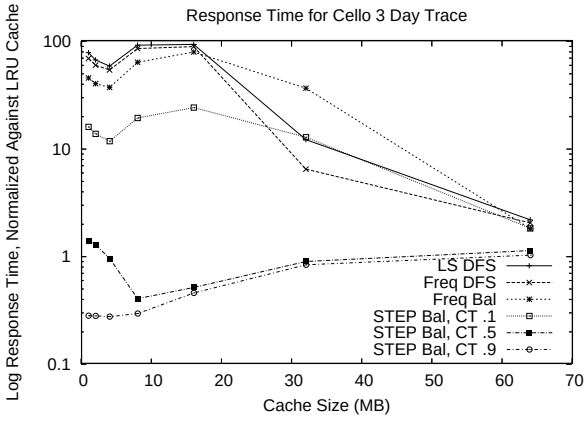
portantly we can see from Figure 5 that these hit rate improvements were generally achieved at the expense of large increases in the total number of I/Os performed. For both the cello and hplajw this could be as significant as a doubling of total I/O operations performed. The sole exceptions to this harsh penalty is the implementations of STEP that use a moderate to high confidence threshold. Basically any moderate implementation of STEP manages to improve hit rates, while incurring little to no additional I/O operations. This implies that STEP is capable of accurately estimating the likelihood of block accesses, with little to no error in either identifying a likely successor, or in rating its true likelihood of access.

3.3 Device Speed *vs.* Power

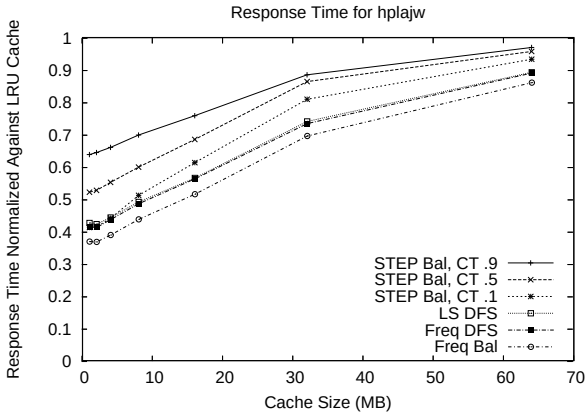
While hit ratios offer a convenient measure of cache performance, and disk read ratios offer a realistic view of the potential added overheads of prefetching caches, the ultimate goal of STEP is to improve performance while conserving device energy consumption. While the total increase in I/O behavior (or ideally, the lack thereof) shown through the disk read ratio metric is indicative of performance in both these areas, it is not a direct relationship. While more I/Os imply greater disk activity, and a higher likelihood of delays to demand fetches, it does not indicate if such delays actually happened for a particular workload. Similarly, disk power consumption is not a direct function of the amount of disk activity, but is also sensitive to the timing of activity due to disk spin-up costs [14, 18]. For this reason Figures 7 and 9 compare the simulated disk response times and energy usage for various cache sizes. Again these values are normalized, with the actual range of values for the LRU cache shown in the caption of each subfigure. For Figure 7, these values represent response time in seconds, but for Figure 9 the actual value represents the percentage of power consumed compared to a continuously running disk.

In Figure 7(a) we see the most dramatic effects of prefetcher overheads. All but the moderate STEP prefetchers suffer tremendous delays compared to a basic LRU cache. This is to be expected, as this is the workload with the high rate of I/Os, and hence any additional or unnecessary prefetch requests will almost certainly delay demand fetches that will follow immediately afterwards. STEP is particularly impressive under these adverse conditions, for it not only avoided delays, but generally showed an improvement in response time for the cache sizes that showed it to increase hit rates. As the cache sizes increase we see a reduction in the negative impact of other prefetchers, and the positive impact of the moderate versions of STEP. This is again due to the cache size growing in relation to the data space and limiting the opportunities to actually prefetch new data. For the lighter hplajw workload (Figure 7(b)), all the prefetchers managed to achieve between 40% to 5% reductions in average response time compared to a simple LRU cache.

A successful predictor will likely impact both the disk energy consumption as well as that of the network interface, basically any devices that would be required to retrieve out-of-cache data. In spite of this we look only to the power consumption of the disk, as this requires the fewest assumptions about the specific nature of the environment. For the network interface there can be a great deal of variation in the effects of networking protocols and the quality of connections, the effects of which are difficult to simulate with a high degree of accuracy and realism. Limiting our energy analysis to a disk device gives us a conservative measure of the benefits of our approach, in a more accurately evaluated environment. In our experiments, power consumption was evaluated by simulating a disk spin-down algorithm with an energy spin-up cost equivalent



(a) cello



(b) hplajw

Figure 7: Response Time Comparison

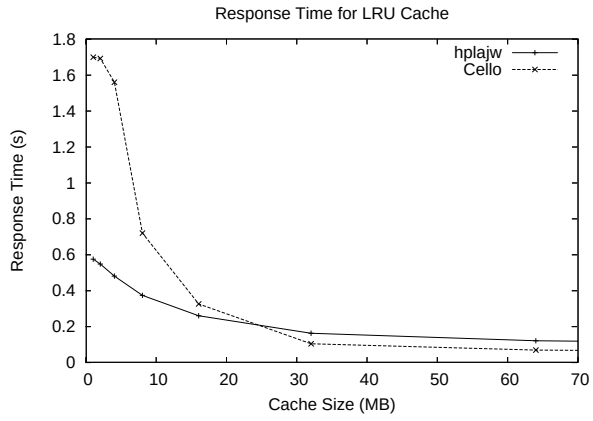
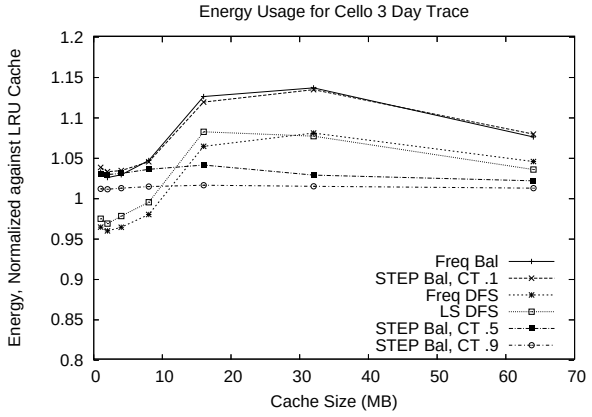
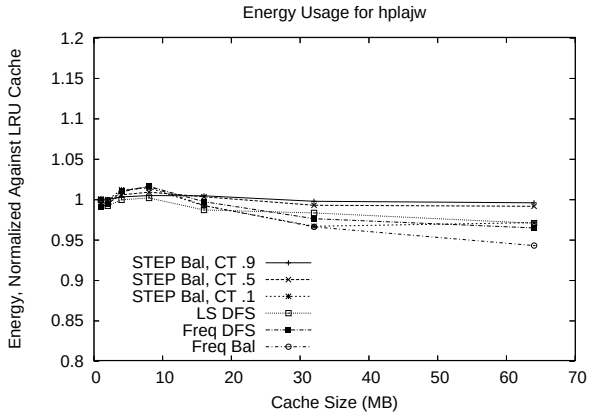


Figure 8: LRU Response Time (the baseline for Figure 7)



(a) cello



(b) hplajw

Figure 9: Energy Usage Comparison

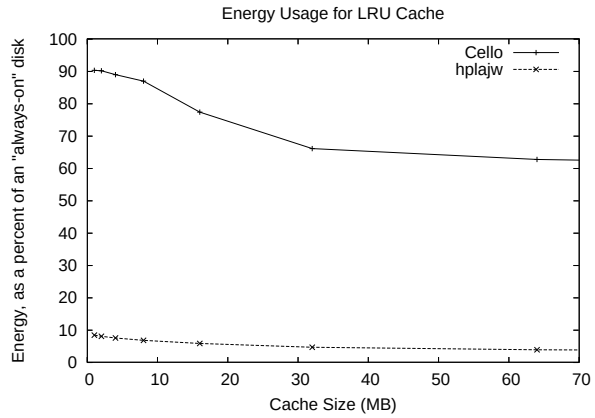


Figure 10: LRU Energy Usage (the baseline for Figure 9)

to 200 seconds of operation. Typical values for this measure range from 100 to 300 seconds [18]. We simulated multiple values for disk spin-down timeout (1 to 90 seconds), and different time penalties (delays) for restarting a spun-down disk (1, 3, and 5 seconds). The results we present are the average performance across these settings.

Figure 9 demonstrates that the addition of prefetchers can have a slightly beneficial or harmful effect on device power consumption compared to a non-prefetching cache. Nonetheless, STEP demontrates the least fluctutation in energy performance, remaining as close as possible to the performance of an LRU cache that does not prefetch. It is important to note that this is the same algorithm and workloads that demonstrated a marked improvement in response time, and yet we can see from this figure that this was achieved with almost no impact on power consumption and hence STEP is energy safe.

4. RELATED WORK

STEP offers a general purpose access prediction and cache prefetching mechanism. We have tested our algorithm against the most general class of storage access, block-level device references, and simulated its behavior as a cache prefetching scheme. There are definitely other approaches to improving the management of mobile and distributed data. These include mobile hoarding, which attempts to pre-identify data that is likely to be accessed during periods of disconnected operation [13, 25], but does not necessarily attempt to improve or reduce local storage device accesses. Other approaches also include those that are application-specific; like web prefetching proxies that can make use of file content for successor hints [8], or explicit compiler generated I/O hints [19]. With STEP, it was not our intent to construct a file hoarding mechanism, or to attempt to utilize any information beyond the access sequence itself, giving us a truly general self-tuning prediction engine that has as little restriction as possible on its wider applications. This wide applicability is particularly aided by STEP's use of a small set of universally observable workload criteria.

Our prefetcher's use of graphs is similar to that originally proposed by Griffioen and Appleton [11]. While Griffioen and Appleton limited the use of these graphs to tracking frequency of access within a particular "look-ahead" window size, the aggregating cache [1] was primarily based on immediate recency (succession) instead of the heuristic of adjacent accesses implied by a sliding window. The use of a last successor model for file prediction, and more elaborate techniques based on pattern matching, were first presented by Lei and Duchamp [17]. Later work by Kroeger and Long [15] compared the predictive performance of the last successor model to Griffioen and Appleton's scheme, and more effective schemes [16] based on context modeling and data compression [26, 6].

Kroeger and Long's implementation of a predictive file system was one of the first works to note the practical and negative impact of predictors that suffer from excessive false predictions, and not enough timely and accurate predictors [16]. Their solution was to modify their compression-based predictor to make further-looking predictions thereby allowing a timely prefetch or the preemption of prefetching workloads by demand workloads. The most recent work on data access prediction attacked this problem by allowing predictors to decline making a prediction when there is a (parametrized) level of doubt [2, 29, 4].

While the predictor proposed by Whittle *et al* was a composite pre-

dictor that selected the best policy among a set [29], we should emphasize that STEP does not operate by discretely selecting among a set of full prediction algorithms, it is in fact based on a set of the simplest heuristics. Of these heuristics, the Null Predictor was originally proposed as part of a multi-expert prediction scheme that simply treated each possible successor as an independent expert [4]. STEP goes well beyond a simple choice, forming a self-optimizing mechanism that estimates the actual likelihood of future access for previously observed successors.

One of the earliest studies of modeling power consumption for computer hard disks was presented by [10], while a more detailed approach was recently presented by [31]. John Wilkes first suggested the use of predictive techniques to dynamically adjust disk spin-downs for improved power conservation [30], while Douglis *et al* demonstrated that perfect non-invasive spin-downs could reduce disk power consumption by up to 60%. They also showed that an on-line algorithm could reduce power consumption by 53% compared to the manufacturer's recommended five minute timeout [7]. These works were primarily concerned with reducing the power consumption of the disk, leaving the negative impact of such techniques on perceived disk speed unanswered.

More recently, prediction has been suggested as a possible means to modify an I/O workload to reduce power consumption in mobile devices [9, 18]. These suggestions focused on the ability of prefetching data to allow for increased idle-time periods, which in turn would hopefully allow greater opportunities for spin-downs. On a similar note, more recent work attempts to actively modify the workload and increase workload burstiness to increase opportunities for disk spin-down, and hopefully reduce the power consumption without adversely affecting disk performance [28, 20, 21, 5]. STEP achieves this goal while automatically adapting its predictors to the workload at hand, as we've demonstrated above using two very disparate device workloads. STEP also does not require any API modifications for device usage, as is required by Weissel *et al* [28].

5. CONCLUSION

Data access prediction has been proposed as a mechanism to overcome latency lag, and more recently as a means of conserving energy in mobile systems. We have presented a fully adaptive predictor, called STEP, that can optimize itself for any arbitrary workload, while simultaneously offering simple adjustment of goals between energy conservation and latency reduction. In other words, we have proposed a predictive prefetching scheme that combines the most generic and useful criteria to enable data prefetching while being energy safe.

Our algorithm, STEP, achieves power savings on mobile computers by eliminating more data fetches, which would otherwise have caused excess energy to be consumed in accessing local storage devices or using the wireless interface to fetch remote data. We have demonstrated our algorithm to perform as well as some of the best access predictors, while incurring almost none of the associated increase in I/O workloads typical of their use. More precisely, our algorithm reduced average response times by approximately 50% compared to an LRU cache, while requiring less than half the I/O operations that traditional predictors would require to achieve the same performance, thereby incurring no energy penalty. In conclusion, our empirical results show that we have been able to achieve the promised performance gains of competing prefetchers, with little to none of the associated costs in terms of power consumption and increased I/O overheads. STEP offers both a highly precise and effective access prediction mechanism, that nonetheless does not unnecessarily increase the total number of I/O operations performed, or the energy consumed by the storage device.

For future work, we intend to experiment with varying threshold values and the effects of such adjustment on predictor performance and adaptability. We also plan to incorporate STEP into a prototype implementation for benchmarking studies.

6. ACKNOWLEDGMENTS

We would like to thank our colleagues at the University of Pittsburgh, particularly within the Advanced Data Management Technologies, the Storage Research, and the Power-Aware Real-Time Systems groups, for their continuous support and assistance. We would further like to thank the members of the Storage Systems Research Center at the University of California, Santa Cruz for valuable discussions, and would also like to thank John Wilkes and the Storage Systems Program at Hewlett Packard Labs for making their traces available.

7. REFERENCES

[1] A. Amer, D. D. E. Long, and R. C. Burns. Group-based management of distributed file caches. In *Proc. of the 22nd Int'l. Conf. on Distributed Computing Systems*, pages 525–534, Vienna, Austria, 2002.

[2] A. Amer, D. D. E. Long, J.-F. Paris, and R. C. Burns. File access prediction with adjustable accuracy. In *Proc. of the 20th IEEE Int'l. Performance, Computing and Communications Conf.*, pages 131–140, Phoenix, AZ, 2002.

[3] A. M. Amer. *Predictive Data Grouping Using Successor Prediction*. PhD thesis, University of California at Santa Cruz, 2002.

[4] K. S. Brandt, D. D. E. Long, and A. Amer. Predicting when not to predict. In *Proc. of the 12th Annual Meeting of the IEEE / ACM Int'l. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 419–426, Volendam, The Netherlands, 2004.

[5] M. Craven and A. Amer. Predictive Reduction of Power and Latency (PuRPLe). In *Proc. of the 22'nd IEEE / 13'th NASA Goddard Conference on Mass Storage Systems and Technologies conference* , Monterey, CA, 2005. (to appear).

[6] K. M. Curewitz, P. Krishnan, and J. S. Vitter. Practical prefetching via data compression. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 257–266, Washington, D. C., 1993.

[7] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the power-hungry disk. In *Proc. of the 1994 Winter USENIX Conf.*, pages 293–306, Boston, MA, 1994.

[8] D. Duchamp. Prefetching hyperlinks. In *Proc. of the Second Usenix Symp. on Internet Technologies and Systems*, pages 127–138, Boulder, CO, 1999.

[9] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proc. of the Symposium on Operating Systems Principles*, pages 48–63, Charleston, SC, 1999.

[10] P. M. Greenawalt. Modeling power management for hard disks. In *Proc. of the Second Int'l. Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 62–66, Durham, NC, 1994.

[11] J. Griffioen and R. Appleton. Reducing file system latency using a predictive approach. In *Proc. of the USENIX Summer Technical Conference*, pages 197–207, Boston, MA, 1994.

[12] M. Herbster and M. K. Warmuth. Tracking the best expert. In *Proc. of the Twelfth Int'l. Conf. on Machine Learning*, pages 286–294, Tahoe City, CA, 1995.

[13] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. In *Proc. of the 13'th ACM Symp. on Operating Systems Principles*, pages 213–225, Pacific Grove, CA, 1991.

[14] P. Krishnan, P. M. Long, and J. S. Vitter. Adaptive disk spindown via optimal rent-to-buy in probabilistic environments. In *Proc. of the Twelfth Int'l. Conf. on Machine Learning*, pages 322–330, Tahoe City, CA, 1995.

[15] T. M. Kroeger and D. D. E. Long. The case for efficient file access pattern modeling. In *Proc. of the Seventh Workshop on Hot Topics in Operating Systems*, pages 14–19, Rio Rico, AZ, 1999.

[16] T. M. Kroeger and D. D. E. Long. Design and implementation of a predictive file prefetching algorithm. In *Proc. of the 2001 USENIX Annual Technical Conf.*, pages 105–118, Boston, MA, 2001.

[17] H. Lei and D. Duchamp. An analytical approach to file prefetching. In *Proc. of the 1997 USENIX Annual Technical Conf.*, pages 275–288, Anaheim, CA, 1997.

[18] J. R. Lorch and A. J. Smith. Software strategies for portable computer energy management. *IEEE Personal Communications Magazine*, 5(3):60–73, 1998.

[19] T. C. Mowry, A. K. Demke, and O. Krieger. Automatic compiler-inserted I/O prefetching for out-of-core applications. In *Proc. of the 1996 Symp. on Operating Systems Design and Implementation*, pages 3–18, Seattle, WA, 1996.

[20] A. E. Papathanasiou and M. L. Scott. Increasing file system burstiness for energy efficiency. Work In Progress at the Symp. on Operating Systems Design and Implementation, Boston, MA, 2002.

[21] A. E. Papathanasiou and M. L. Scott. Energy efficient prefetching and caching. In *Proc. of the USENIX 2004 Annual Technical Conferfence*, pages 255–268, Boston, MA, 2004.

[22] C. Ruemmler and J. Wilkes. UNIX disk access patterns. In *Proc. of the 1993 Winter USENIX Technical Conf.*, pages 405–420, San Diego, CA, 1993.

[23] G. Santhanakrishnan, A. Amer, and P. K. Chrysanthis. A Goal-Oriented Self-Tuning caching algorithm. In *Proc. of the 3'rd IEEE International Performance Computing and Communications Conference*, pages 311–312, Phoeniz, AZ, 2004.

[24] G. Santhanakrishnan, A. Amer, P. K. Chrysanthis, and D. Li. GD-GhOST: A Goal-Oriented Self-Tuning caching algorithm. In *Proc. of the 19'th ACM Symp. on Applied Computing*, pages 1141–1145, Nicosia, Cyprus, 2004.

[25] Y. Saygin, O. Ulusoy, and A. K. Elmagarmid. Association rules for supporting hoarding in mobile computing environments. In *Proc. of the 10'th Int'l. Workshop on Research Issues in Data Engineering*, pages 71–78, San Diego, CA, 2000.

[26] J. S. Vitter and P. Krishnan. Optimal prefetching via data compression. *JACM*, 43(5):771–793, 1996.

[27] M. Warmuth and N. Littlestone. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[28] A. Weissel, B. Beutel, and F. Bellosa. Cooperative I/O: A novel I/O semantics for energy-aware applications. In *Proc. of the 2002 Symp. on Operating Systems Design and Implementation*, pages 117–130, Boston, MA, 2002.

[29] G. A. S. Whittle, J.-F. Paris, A. Amer, D. D. E. Long, and R. Burns. Using multiple predictors to improve the accuracy of file access predictions. In *Proc. of the 20'th IEEE/11'th NASA Goddard Conf. on Mass Storage Systems and Technologies*, pages 230–240, San Diego,CA, 2003.

[30] J. Wilkes. Predictive power conservation. Technical Report HPL-CSP-92-5, Hewlett-Packard Laboratories, Palo Alto, CA, 1992.

[31] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modeling hard-disk power consumption. In *Proc. of the 2003 Usenix Conf. on File and Storage Technologies*, pages 217–230, San Francisco, CA, 2003.