Appears in *Proceedings of the ACM SIGMOD Conference*, San Jose, CA, May 1995

Broadcast Disks: Data Management for Asymmetric Communication Environments

Swarup Acharya
Brown University
sa@cs.brown.edu

Rafael Alonso
MITL
alonso@mitl.research.panasonic.com

Michael Franklin
University of Maryland
franklin@cs.umd.edu

Stanley Zdonik
Brown University
sbz@cs.brown.edu

Abstract

This paper proposes the use of repetitive broadcast as a way of augmenting the memory hierarchy of clients in an asymmetric communication environment. We describe a new technique called "Broadcast Disks" for structuring the broadcast in a way that provides improved performance for non-uniformly accessed data. The Broadcast Disk superimposes multiple disks spinning at different speeds on a single broadcast channel — in effect creating an arbitrarily fine-grained memory hierarchy. In addition to proposing and defining the mechanism, a main result of this work is that exploiting the potential of the broadcast structure requires a re-evaluation of basic cache management policies. We examine several "pure" cache management policies and develop and measure implementable approximations to these policies. These results and others are presented in a set of simulation studies that substantiates the basic idea and develops some of the intuitions required to design a particular broadcast program.

1 Introduction

1.1 Asymmetric Communication Environments

In many existing and emerging application domains the *downstream* communication capacity from servers to clients is much greater than the *upstream* communication capacity from clients back to servers. For example, in a wireless mobile network, servers may have a relatively high bandwidth broadcast capability while clients cannot transmit or can do so only over a lower bandwidth (e.g., cellular) link. Such systems have been proposed for many application domains, including traffic information systems, hospital information systems, public safety applications, and wireless classrooms (e.g.,[Katz94, Imie94a]). We refer to these environments as *Asymmetric Communications Environments*.

Communications asymmetry can arise in two ways: the first is from the bandwidth limitations of the physical communications medium. An example of physical asymmetry is the wireless environment as described above; stationary servers have powerful broadcast transmitters while mobile clients have little or no transmission capability. Perhaps less obviously, communications asymmetry can also arise from the patterns of *information flow* in the application. For example, an information retrieval system in which the number of clients is far greater than the number of servers is asymmetric because there is insufficient capacity (either in the network or at the servers) to handle the simultaneous requests generated by the multiple clients.

Because asymmetry can arise due to either physical devices or workload characteristics, the class of asymmetric communications environments spans a wide range of important systems and applications, encompassing both wired and wireless networks. Examples include:

- Wireless networks with stationary base stations and mobile clients.
- Information dispersal systems for volatile, time-sensitive information such as stock prices, weather information, traffic updates, factory floor information, etc.
- Cable or satellite broadcast television networks with set-top boxes that allow viewers to communicate with the broadcasting home office, and video-on-demand servers.
- Information retrieval systems with large client populations, such as mail-order catalog services, mutual fund information services, software help desks, etc.

1.2 Broadcast Disks

In traditional client-server information systems, clients initiate data transfers by sending requests to a server. Such systems are *pull-based*; the clients "pull" data from the server in order to provide data to locally running applications. Pull-based systems are a poor match for asymmetric communications environments, as they require substantial upstream communications capabilities. To address this incompatibility, we have proposed a new information system architecture that exploits the relative abundance of downstream communication capacity in asymmetric environments. This new architecture is called *Broadcast Disks*. The central idea is that the servers exploit their advantage in bandwidth by *broadcasting* data to multiple clients. We refer to this arrangement as a *push-based* architecture; data is pushed from the server out to the clients.

In this approach, a server continuously and repeatedly broadcasts data to the clients. In effect, the broadcast channel becomes a "disk" from which clients can retrieve data as it goes by. Broadcasting data has been proposed previously [Herm87, Giff90, Imie94b]. Our technique differs, however, in two ways. First, we superimpose *multiple disks* of different sizes and speeds on the broadcast medium, in effect, creating an arbitrarily fine-grained memory hierarchy. Second, we exploit client storage resources as an integral part of this extended memory hierarchy.

The broadcast is created by assigning data items to different "disks" of varying sizes and speeds, and then multiplexing the disks on the broadcast channel. Items stored on faster disks are broadcast more often than items on slower disks. This approach creates a memory hierarchy in which data on the fast disks are "closer" to the clients than data on slower disks. The number of disks, their sizes, and relative speeds can be adjusted, in order to more closely match the broadcast with the desired access probabilities at the clients. If the server has an indication of the client access patterns (e.g., by watching their previous activity or from a description of intended future use from each client), then hot pages (i.e., those that are more likely to be of interest to a larger part of the client community) can be brought closer while cold pages can be pushed further away.

1.3 Scope of the Paper

In this paper, we focus on a restricted broadcast environment in order to make an initial study of the broadcast disk approach feasible. The restrictions include:

- The client population and their access patterns do not change. This implies that the content and the organization of the broadcast program remains static.
- Data is read-only; there are no updates either by the clients or at the servers.
- Clients retrieve data items from the broadcast on demand; there is no prefetching.
- Clients make no use of their upstream communications capability, i.e., they provide no feedback to servers.

In this environment, there are two main interrelated issues that must be addressed:

1. Given a client population and a specification of their access probabilities for the data items, how does the server construct a broadcast program to satisfy the needs of the clients?
2. Given that the server has chosen a particular broadcast program, how does each client manage its local data cache to maximize its own performance?

The remainder of the paper is organized as follows. Section 2 discusses the way in which we structure the broadcast program and Section 3 shows how the client's cache management policy should be designed to complement this choice. Section 4 describes our simulation model and Section 5 develops the main experimental results derived from this model. Section 6 compares our work to previous work on repetitive broadcast. Section 7 summarizes our results and describes our future work.

2 Structuring the Broadcast Disk

2.1 Properties of Broadcast Programs

In a push-based information system, the server must construct a broadcast "program" to meet the needs of the client population. In the simplest scenario, given an indication of the data items that are desired by each client listening to the broadcast, the server would simply take the union of the requests and broadcast the resulting set of data items cyclicly. Such a broadcast is depicted in Figure 1.

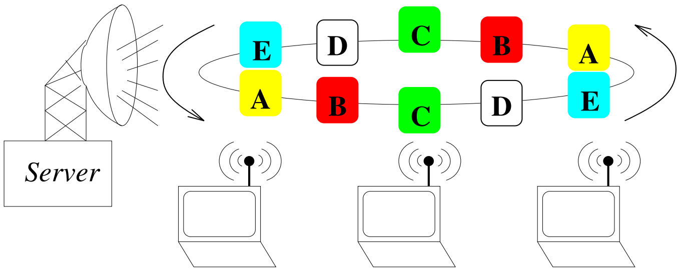

Figure 1: A Flat Broadcast Program

When an application running on a client needs a data item, it first attempts to retrieve that item from the local memory or disk. If the desired item is not found, then the client monitors the broadcast and waits for the item to arrive.[1] With the flat broadcast, the expected wait for an item on the broadcast is the same for all items (namely, half a broadcast period) regardless of their relative importance to the clients. This "flat" approach has been adopted in earlier work on broadcast-based database systems such as Datacycle[Bowe92] and [Imie94a].

Alternatively, the server can broadcast different items with differing frequency. Such a broadcast program can emphasize the most popular items and de-emphasize the less popular ones. Theoretically, the generation of such non-flat broadcast programs can be addressed as a *bandwidth allocation* problem; given all of the client access probabilities, the server determines the optimal percentage of the broadcast bandwidth that should be allocated to each item. The broadcast program can then be generated randomly according to those bandwidth allocations, such that the *average* inter-arrival time between two instances of the same item matches the needs of the client population. However, such a random broadcast will not be optimal in terms of minimizing expected delay due to the variance in the inter-arrival times.

A simple example demonstrating these points is shown in Figure 2. The figure shows three different broadcast programs for a data set containing three equal-length items (e.g., pages). Program (a) is a flat broadcast, while (b) and (c) both broadcast page `A` twice as often as pages `B` and `C`. Program (b) is a *skewed* broadcast, in which subsequent broadcasts of page `A` are clustered together. In contrast, program (c) is *regular*; there is no variance in the inter-arrival time for each page. The performance characteristics of program (c) are the same as if page `A` was stored on a disk that is spinning twice as fast as the disk containing pages `B` and `C`. Thus, we refer to program (c) as a *Multi-disk* broadcast.

[1] This discussion assumes that broadcast items are self-identifying. Another option is to provide an index, as is discussed in [Imie94b].

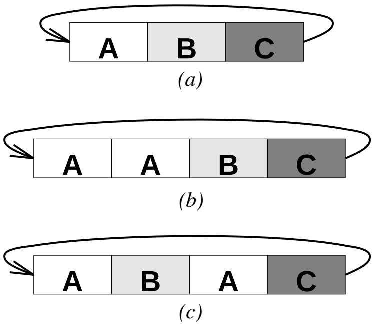

Figure 2: Three Example Broadcast Programs

Access Probability			Expected Delay (in broadcast units)		
A	**B**	**C**	**Flat** (a)	**Skewed** (b)	**Multi-disk** (c)
0.333	0.333	0.333	1.50	1.75	1.67
0.50	0.25	0.25	1.50	1.63	1.50
0.75	0.125	0.125	1.50	1.44	1.25
0.90	0.05	0.05	1.50	1.33	1.10
1.0	0.0	0.0	1.50	1.25	1.00

Table 1: Expected Delay For Various Access Probabilities

Table 1 shows the expected delay for page requests given various client access probability distributions, for the three different broadcast programs. The expected delay is calculated by multiplying the probability of access for each page times the expected delay for that page and summing the results. There are three major points that are demonstrated by this table. The first point is that for uniform page access probabilities (1/3 each), a flat disk has the best expected performance. This fact demonstrates a fundamental constraint of the Broadcast Disk paradigm, namely, that due to fixed bandwidth, *increasing the broadcast rate of one item must necessarily decrease the broadcast rate of one or more other items*. The second point, however, is that as the access probabilities become increasingly skewed, the non-flat programs perform increasingly better.

The third point demonstrated by Table 1 is that the Multi-disk program always performs better than the skewed program. This behavior is the result of the so-called Bus Stop Paradox. If the inter-arrival rate (i.e., broadcast rate) of a page is fixed, then the expected delay for a request arriving at a random time is one-half of the gap between successive broadcasts of the page. In contrast, if there is variance in the inter-arrival rate, then the gaps between broadcasts will be of different lengths. In this case, the probability of a request arriving during a large gap is greater than the probability of the request arriving during a short gap. Thus the expected delay increases as the variance in inter-arrival rate increases.

In addition to performance benefits, a Multi-disk broadcast has several other advantages over a random (skewed) broadcast program. First, the randomness in arrivals can reduce the effectiveness of some prefetching techniques that require knowledge of exactly when a particular item will next be broadcast [Zdon94]. Second, the randomness of broadcast disallows the use of "sleeping" to reduce power consumption (as in [Imie94b]). Finally, there is no notion of "period" for such a broadcast. Periodicity may be important for providing correct semantics for updates (e.g., as was done in Datacycle [Herm87, Bowe92]) and for introducing changes to the structure of the broadcast program. For these reasons, we argue that a broadcast program should have the following features:

- The inter-arrival times of subsequent copies of a data item should be fixed.
- There should be a well defined unit of broadcast after which the broadcast repeats (i.e., it should be periodic).
- Subject to the above two constraints, as much of the available broadcast bandwidth should be used as possible.

2.2 Broadcast Program Generation

In this section we present a model for describing the structure of broadcast programs and describe an algorithm that generates broadcast programs with the desired features listed in the previous section. The algorithm imposes a Multi-disk structure on the broadcast medium in a way that allows substantial flexibility in fitting the relative broadcast frequencies of data items to the access probabilities of a client population.

The algorithm has the following steps (for simplicity, assume that data items are "pages", that is, they are of a uniform, fixed length):

1. *Order the pages from hottest (most popular) to coldest.*
2. *Partition the list of pages into multiple ranges*, where each range contains pages with similar access probabilities. These ranges are referred to as *disks*.
3. *Choose the relative frequency of broadcast for each of the disks.* The only restriction on the relative frequencies is that they must be integers. For example given two disks, disk 1 could be broadcast three times for every two times that disk 2 is broadcast, thus, *rel_freq(1)* = 3, and *rel_freq(2)* = 2.
4. *Split each disk into a number of smaller units.* These units are called *chunks* (C_{ij} refers to the j^{th} chunk in disk i). First, calculate *max_chunks* as the Least Common Multiple (LCM) of the relative frequencies. Then, split each disk i into *num_chunks(i) = max_chunks / rel_freq(i)* chunks. In the previous example, *num_chunks(1)* would be 2, while *num_chunks(2)* would be 3.
5. *Create the broadcast program* by interleaving the chunks of each disk in the following manner:

```
for i := 0 to max_chunks - 1
    for j := 1 to num_disks
        Broadcast chunk C_{j,(i mod num_chunks(j))}
    endfor
endfor
```

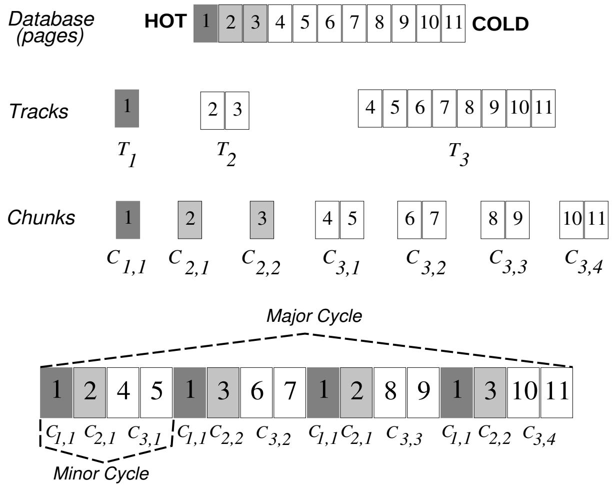

Figure 3: Deriving a Server Broadcast Program

Figure 3 shows an example of broadcast program generation. Assume a list of pages that has been partitioned into three disks, in which pages in disk 1 are to be broadcast twice as frequently as pages in disk 2, and four times as frequently as pages in disk 3. Therefore, *rel_freq(1)* = 4, *rel_freq(2)* = 2, and *rel_freq(3)* = 1. These disks are split into chunks according to step 4 of the algorithm. That is *max_chunks* is 4, so *num_chunks(1)* = 1, *num_chunks(2)* = 2, and *num_chunks(3)* = 4. Note that the chunks of different disks can be of differing sizes. The resulting broadcast consists of 4 *minor cycles* (containing one chunk of each disk) which is the LCM of the relative frequencies. The resulting broadcast has a period of 16 pages. This broadcast produces a three-level memory hierarchy in which disk one is the smallest and fastest level and disk three is the largest and slowest level. Thus, the multi-level broadcast corresponds to the traditional notion of a memory hierarchy.

The algorithm produces a periodic broadcast program with fixed inter-arrival times per page. Some broadcast slots may be unused however, if it is not possible to evenly divide a disk into the required number of chunks (i.e., in Step 4 of the algorithm). Of course, such extra slots need not be wasted, they can be used to broadcast additional information such as indexes, updates, or invalidations; or even for extra broadcasts of extremely important pages. Furthermore, it is anticipated that the number of disks will be small (on the order of 2 to 5) and the number of pages to be broadcast will be substantially larger, so that unused slots (if any) will be only a small fraction of the total number of slots; also, the relative frequencies can be adjusted slightly to reduce the number of unused slots, if necessary.

The disk model, while being fairly simple, allows for the creation of broadcast programs that can be fine-tuned to support a particular access probability distribution. There are three inter-related types of knobs that can be turned to vary the shape of the broadcast. First, the number of disks (*num_disks*) determines the number of different frequencies with which pages will be broadcast. Then, for each disk, the number of pages per disk, and its relative frequency of broadcast (*rel_freq(i)*) determine the size of the broadcast, and hence the arrival rate (in real, rather than relative time) for pages on each disk. For example, adding a page to a fast disk can significantly increase the delay for pages on the slower disks. Intuitively, we expect that fast disks will be configured to have many fewer pages than the slower disks, although our model does not enforce this constraint.

Recall that the only constraint on the relative broadcast frequencies of the disks is that they be expressed as positive integers. Thus, it is possible to have arbitrarily fine distinctions in broadcasts such as a disk that rotates 141 times for every 98 times a slower disk rotates. However, this ratio results in a broadcast that has a very long period (i.e., nearly 14,000 rotations of the fast disk). Furthermore, this requires that the slower disk be of a size that can be split into 141 fairly equal chunks. In addition, it is unlikely that such fine tuning will produce any significant performance benefit (i.e., compared to a 3 to 2 ratio). Therefore, in practice, relative frequencies should be chosen with care and when possible, approximated to simpler ratios.

While the algorithm specified above generates broadcast programs with the properties that we desire, it does not help in the selection of the various parameter values that shape the broadcast. The automatic determination of these parameters for a given access probability distribution is a very interesting optimization problem, and is one focus of our on-going work. This issue is beyond the scope of the current paper, however. In this paper we focus on examining the basic properties of this new paradigm of broadcast disks. The broadcast disk changes many basic assumptions on which traditional pull-based memory hierarchies are founded. As a result, it is imperative to first develop an understanding of the fundamental tradeoffs that affect the performance of a broadcast system. The performance study described in Section 5 presents an initial investigation of these issues.

3 Client Cache Management

The shared nature of the broadcast disk, while in principle allowing for nearly unlimited scalability, in fact gives rise to a fundamental tradeoff: *tuning the performance of the broadcast is a zero-sum game*; improving the broadcast for any one access probability distribution will hurt the performance of clients with different access distributions. The way out of this dilemma is to exploit the local memory and/or disk of the client machines to cache pages obtained from the broadcast. This observation leads to a novel and important result of this work: namely, that the introduction of broadcast *fundamentally changes the role of client caching* in a client-server information system. In traditional, pull-based systems clients cache their *hottest* data (i.e., the items that they are most likely to access in the future). In the push-based environment, this use of the cache can lead to poor performance if the server's broadcast is poorly matched to the client's page access distribution. This difference arises because of the serial nature of the broadcast disk — broadcast pages are *not* all equidistant from the client.

If the server can tailor the broadcast program to the needs of a particular client, then the client can simply cache its hottest pages. Once the client has loaded the hottest pages in its cache, then the server can place those pages on a slower spinning disk. This frees up valuable space in the fastest spinning disks for additional pages. In general, however, there are several factors that could cause the server's broadcast to be sub-optimal for a particular client:

- The access distribution that the client gives the server may be inaccurate.
- A client's access distribution may change over time.
- The server may give higher priority to the needs of other clients with different access distributions.
- The server may have to average its broadcast over the needs of a large client population. Such a broadcast program is likely to be sub-optimal from the point of view of any one client.

For these reasons, in a push-based system clients must use their cache not to store simply their hottest pages, but rather, to store *those pages for which the local probability of access is significantly greater than the page's frequency of broadcast.* For example, if there is a page P that is accessed frequently only by client C and no other clients, then that page is likely to be broadcast on a slow disk. To avoid long waits for the page, client C must keep page P cached locally. In contrast, a page Q that is accessed frequently by most clients (including client C), will be broadcast on a very fast disk, reducing the value of caching it.

The above argument leads to the need for *cost-based page replacement*. That is, the cost of obtaining a page on a cache miss must be accounted for during page replacement decisions. A standard page replacement policy tries to replace the cache-resident page with the lowest probability of access (e.g., this is what LRU tries to approximate). It can be shown that under certain assumptions, an optimal replacement strategy is one that replaces the cache-resident page having the lowest ratio between its probability of access (*P*) and its frequency of broadcast (*X*). We refer to this ratio (P/X) as $\mathcal{PIX}$ (P Inverse X). As an example of the use of $\mathcal{PIX}$, consider two pages. One page is accessed 1% of the time at a particular client and is also broadcast 1% of the time. A second page is accessed only 0.5% of the time at the client, but is broadcast only 0.1% of the time. In this example, the former page has a lower $\mathcal{PIX}$ value than the latter. As a result, a page replacement policy based on $\mathcal{PIX}$ would replace the first page in favor of the second, even though the first page is accessed twice as frequently.

While $\mathcal{PIX}$ can be shown to be an optimal policy under certain conditions, it is not a practical policy to implement because it requires: 1) perfect knowledge of access probabilities and 2) comparison of $\mathcal{PIX}$ values for *all* cache-resident pages at page replacement time. For this reason we have investigated implementable cost-based algorithms that are intended to approximate the performance of $\mathcal{PIX}$. One such algorithm, adds frequency of broadcast to an LRU-style policy. This new policy is called $\mathcal{LIX}$ and is described and analyzed in Section 5.4.

4 Modeling the Broadcast Environment

In order to better understand the properties of broadcast program generation and client cache management we have constructed a simulation model of the broadcast disk environment. The simulator, which is implemented using CSIM [Schw86], models a single server that continuously broadcasts pages and a single client that continuously accesses pages from the broadcast and from its cache. In the simulator, the client generates requests for *logical* pages. These logical pages are then mapped to the *physical* pages that are broadcast by the server.

CacheSize	Client cache size (in pages)
ThinkTime	Time between client page accesses (in broadcast units)
AccessRange	# of pages in range accessed by client
θ	Zipf distribution parameter
RegionSize	# of pages per region for Zipf distribution

Table 2: Client Parameter Description

The mapping of *logical* pages to *physical* pages allows the server broadcast to be varied with respect to the client workload. This flexibility allows the simulator to model the impact of a large client population on the performance of a single client, without having to model the other clients. For example, having the client access only a subset of the pages models the fact that the server is broadcasting pages for other clients as well. Furthermore, by systematically perturbing the client's page access probabilities with respect to the server's expectation of those probabilities, we are able to vary the degree to which the server broadcast favors the particular client that we are modeling. The simulation model is described in the following sections.

4.1 Client Execution Model

The parameters that describe the operation of the client are shown in Table 2. The simulator measures performance in logical time units called *broadcast units*. A broadcast unit is the time required to broadcast a single page. In general, the results obtained from the simulator are valid across many possible broadcast media. The actual response times experienced for a given medium will depend on the amount of real time required to broadcast a page.

The client runs a continuous loop that randomly requests a page according to a specified distribution. The client has a cache that can hold *CacheSize* pages. If the requested page is not cache-resident, then the client waits for the page to arrive on the broadcast and then brings the requested page into its cache. Client cache management is done similarly to buffer management in a traditional system; if all cache slots are occupied, then a page replacement policy is used to choose a victim for replacement.[2] Once the requested page is cache resident, the client waits *ThinkTime* broadcast units of time and then makes the next request. The *ThinkTime* parameter allows the cost of client processing relative to page broadcast time to be adjusted, thus it can be used to model workload processing as well as the relative speeds of the CPU and the broadcast medium.

The client chooses the pages to access from the range 0 to *AccessRange*-1, which can be a subset of the pages that are broadcast. All pages outside of this range have a zero probability of access at the client. Within the range the page access probabilities follow a Zipf distribution [Knut81, Gray94], with page 0 being the most frequently accessed, and page *AccessRange*-1 being the least frequently accessed. The Zipf distribution is typically used to model non-uniform access patterns. It produces access patterns that

[2] We discuss the performance of various replacement policies in Section 5.

ServerDBSize	Number of distinct pages to be broadcast
NumDisks	Number of disks
DiskSize$_i$	Size of disk i (in pages)
Δ	Broadcast shape parameter
Offset	Offset from default client access
Noise	% workload deviation

Table 3: Server Parameter Description

become increasingly skewed as θ increases — the probability of accessing any page numbered i is proportional to $(1/i)^\theta$. Similar to earlier models of skewed access [Dan90], we partition the pages into regions of *RegionSize* pages each, such that the probability of accessing any page within a region is uniform; the Zipf distribution is applied to these regions. Regions do not overlap so there are *AccessRange/RegionSize* regions.

4.2 Server Execution Model

The parameters that describe the operation of the server are shown in Table 3. The server broadcasts pages in the range of 0 to *ServerDBSize*-1, where *ServerDBSize* $\geq$ *AccessRange*. These pages are interleaved into a broadcast program according to the algorithm described in Section 2. This program is broadcast repeatedly by the server. The structure of the broadcast program is described by several parameters. *NumDisks* is the number of levels (i.e., "disks") in the multi-disk program. By convention disks are numbered from 1 (fastest) to N=*NumDisks* (slowest). *DiskSize$_i$*, $i \in$ [1..N], is the number of pages assigned to each disk i. Each page is broadcast on exactly one disk, so the sum of *DiskSize$_i$* over all i is equal to the *ServerDBSize*.

In addition to the size and number of disks, the model must also capture their relative speeds. As described in Section 2, the relative speeds of the various disks can be any positive integers. In order to make experimentation tractable, however, we introduce a parameter called Δ, which determines the relative frequencies of the disks in a restricted manner. Using Δ, the frequency of broadcast $rel\_freq(i)$ of each disk i, can be computed relative to $rel\_freq(\mathrm{N})$, the broadcast frequency of the slowest disk (disk N) as follows:

$$\frac{rel\_freq(i)}{rel\_freq(\mathrm{N})} = (\mathrm{N} - \mathrm{i})\Delta + 1$$

When Δ is zero, the broadcast is flat: all disks spin at the same speed. As Δ is increased, the speed differentials among the disks increase. For example, for a 3-disk broadcast, when $\Delta = 1$, disk 1 spins three times as fast as disk 3, while disk 2 spins twice as fast as disk 3. When $\Delta = 3$, the relative speeds are 7, 4, and 1 for disks 1, 2, and 3 respectively. It is important to note that Δ is used in the study only to organize the space of disk configurations that we examine. It is not part of the disk model as described in Section 2.

The remaining two parameters, *Offset* and *Noise*, are used to modify the mapping between the *logical* pages requested by the client and the *physical* pages broadcast by the server. When *Offset* and *Noise* are both set to zero, then the logical to physical mapping is simply the identity function. In this case, the *DiskSize*$_1$ hottest pages from the client's perspective (i.e., 0 to *DiskSize*$_1 - 1$) are placed on disk 1, the next *DiskSize*$_2$ hottest pages are placed on disk 2, etc. However, as discussed

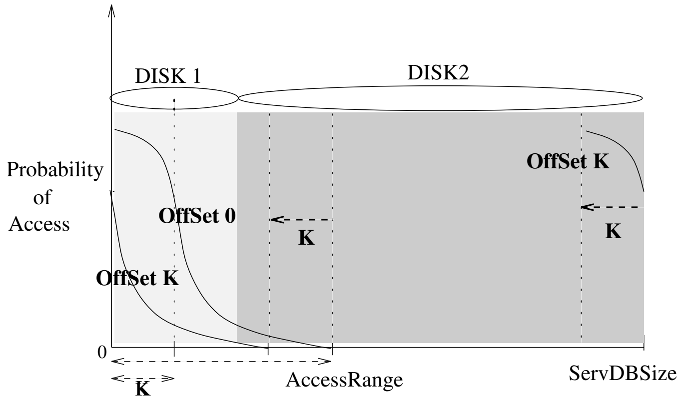

Figure 4: Using Offset to vary client access

in Section 3, this mapping may be sub-optimal due to client caching. Some client cache management policies tend to fix certain pages in the client's buffer, and thus, those pages do not need to be broadcast frequently. In such cases, the best broadcast can be obtained by shifting the hottest pages from the fastest disk to the slowest. *Offset* is the number of pages that are shifted in this manner. An offset of K shifts the access pattern by K pages, pushing the K hottest pages to the end of the slowest disk and bringing colder pages to the faster disks. The use of offset is demonstrated in Figure 4.

In contrast to *Offset*, which is used to provide a better broadcast for the client, the parameter *Noise* is used to introduce disagreement between the needs of the client and the broadcast program generated by the server. As described in Section 2, such disagreement can arise in many ways, including dynamic client access patterns and conflicting access requirements among a population of clients. *Noise* determines the percentage of pages for which there may be a mismatch between the client and the server. That is, with probability *Noise* the mapping of a page may be switched with a different page.

The generation of the server broadcast program works as follows. First, the mapping from logical to physical pages is generated as the identity function. Second, this mapping is shifted by *Offset* pages as described above. Third, for each page in the mapping, a coin weighted by *Noise* is tossed. If based on the coin toss, a page i is selected to be swapped then a disk d is uniformly chosen to be its new destination.[3] To make way for i, an existing page j on d is chosen, and i and j exchange mappings.

5 Experiments and Results

In this section, we use the simulation model to explore the performance characteristics of the broadcast disk. The primary performance metric employed in this study is the response time at the client, measured in *broadcast units*. The server database size (*ServerDBSize*) was 5000 pages, and the client access range *AccessRange* was 1000 pages. The client cache size was varied from 1 (i.e., no caching) to 500 (i.e., half of the access range). We studied several different two-disk and three-disk configurations of broadcast programs. All

[3] Note that a page may be swapped with a page on its own disk. Such a swap does not affect performance in the steady state, so *Noise* represents the upper limit on the number of changes.

ThinkTime	2.0
ServerDBSize	5000
AccessRange	1000
CacheSize	50(5%), 250(25%), 500(50%)
Δ	1,2,. . .7
θ	0.95
Offset	0, *CacheSize*
Noise	0%, 15%, 30%, 45%, 60%, 75%
RegionSize	50

Table 4: Parameter Settings

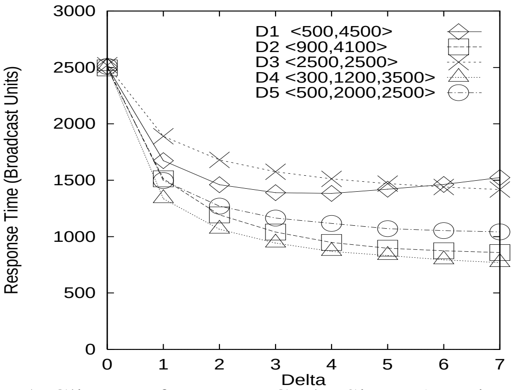

Figure 5: Client Performance, Cache Size = 1, Noise = 0%

of the results presented in the paper were obtained once the client performance reached steady state. The cache warm-up effects were eliminated by beginning our measurements only after the cache was full, and then running the experiment for 15,000 or more client page requests (until steady state).

Table 4 shows the parameter settings used in these experiments. It should be noted that the results described in this section are a very small subset of the results that have been obtained. These results have been chosen because they demonstrate many of the unique performance aspects and tradeoffs of the broadcast disk environment, and because they identify important areas for future study.

5.1 Experiment 1: No Caching, 0% Noise

The first set of results examine the case where the client performs no caching (i.e., it has a cache size of one page). Figure 5 shows the client response time vs. Δ for a number of two and three disk configurations. In this graph, *Noise* is set to 0%, meaning that the server is providing preferential treatment to the client (i.e., it is giving highest priority to this client's pages). As Δ is increased along the x-axis of the figure, the skew in the relative speeds of the disks is increased (as described in Section 4). As shown in the figure, the general trend in these cases is that response time improves with increasing disk skew. When $\Delta = 0$, the broadcast is flat (i.e., all disks rotate at the same speed). In this case, as would be expected, all disks result in a response time of 2500 pages — half the *ServerDBSize*. As Δ is increased, all of the disk configurations shown provide an improvement over the flat disk. The degree of improvement begins to flatten for most configurations around a Δ value of 3 or 4.

Turning to the various disk configurations, we first examine the two-disk configurations: D1, D2, and D3. For D1, 500 pages fit on the first (i.e., fastest) disk. Because *Noise* and *Offset* are both zero, the hottest half of the client's access range is on the fast disk, and the colder half is on the slower disk. As Δ is increased, performance improves until $\Delta = 3$ because the hotter pages are brought closer. Beyond this point, the degradation caused by the access to the slow pages (which get pushed further away) begins to hurt performance. In contrast, D2, which places 90% of the client access range (900 pages) on the fast disk improves with increasing Δ for all values of Δ in this experiment. Because most of the accessed pages are on the fast disk, increasing Δ pushes the colder and unused pages further away, allowing the accessed pages to arrive more frequently. At some point, however, the penalty for slowing down the 10% will become so great that the curve will turn up again as in the previous case. The final two-disk configuration, D3, has equal sized disks. Although all of the accessed data fits on the fast disk, the fast disk also includes many unaccessed pages. The size of the fast disk causes the frequencies of the pages on this disk to be lower than the frequencies of pages on the fast disks of D2 and D1 at corresponding values of Δ. As a result, D3 has the worst performance of the two-disk configurations for most of the Δ values shown.

Turning to the three-disk configurations: D4 and D5, it can be seen that configuration D4, which has a fast disk of 300 pages has the best performance across the entire range. At a Δ of 7, its response time is only one-third of the flat-disk response time. D5, which is simply the D3 disk with its first disk split across two disks, performs better than its two-disk counterpart. The extra level of disk makes it easier to match the broadcast program to the client's needs. However, note that response time for D5 is typically higher than the two-disk D2, and thus, the extra disk level does not necessarily ensure better performance.

5.2 Experiment 2: Noise and No Caching

In the previous experiment, the broadcast program generation was done giving our client's access pattern the highest priority. In this experiment we examine the performance of the broadcast disk as the server shifts its priority away from this client (i.e., as *Noise* is increased). These results are shown in Figures 6 and 7, which show how the client performs in the presence of increasing noise for configurations D3 (two-disks) and D5 (three-disks) from Figure 5 respectively. As expected, performance suffers for both configurations as the *Noise* is increased; as the mismatch between the broadcast and the client's needs increases, the skew in disk speeds starts to hurt performance. Ultimately, if the mismatch becomes great enough, the multi-disk approach can have worse performance than the flat disk. This is shown in the performance disk of D3 (Figure 6). This susceptibility to a broadcast mismatch is to be expected, as the client accesses all of its data from the broadcast channel. Thus, it is clear that if a client does not have a cache, the broadcast must be well suited for that client's access demands in order to gain the benefits of the multi-disk approach.

5.3 Experiment 3: Caching and Noise

The previous experiments showed that even in the absence of caching, a multi-level disk scheme can improve performance, but that without a cache, performance can suffer if the broadcast program is poorly suited to the client's access demands. In this experiment we introduce the use of a

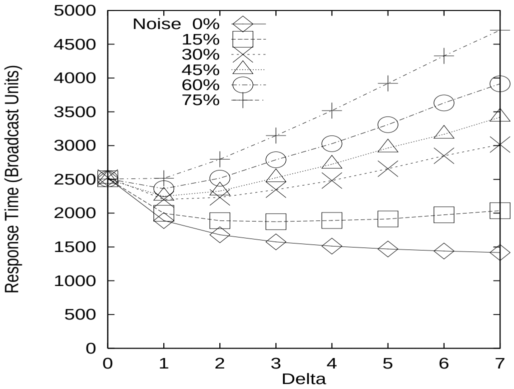

Figure 6: Noise Sensitivity - Disk D3(<2500,2500>)
CacheSize = 1

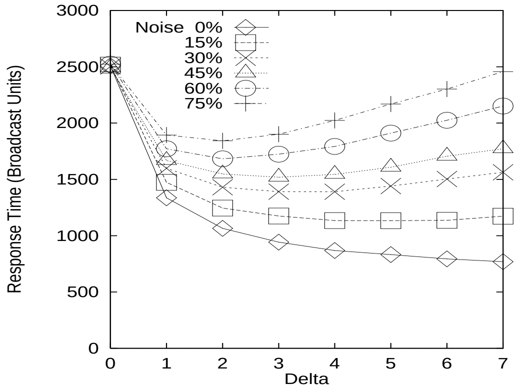

Figure 7: Noise Sensitivity - Disk D5(<300,1200,3500>)
CacheSize = 1

client cache, to reduce the expected page access delay and to increase the client's tolerance to mismatches in the broadcast program. We use an idealized page replacement policy called $\mathcal{P}$, which keeps the pages with the highest probability of access in the cache. $\mathcal{P}$, however, it is not an implementable policy, as it requires perfect knowledge of access probabilities and a great deal of local computation.[4] We use $\mathcal{P}$, therefore, to gain an understanding of the performance in a simplified setting and as a point-of-reference for other (implementable) policies.

In steady state, a client using the $\mathcal{P}$ replacement policy will have the *CacheSize* hottest pages in its cache. Consequently, broadcasting these cache-resident pages on the fastest disk is a waste of bandwidth. Thus, as stated in Section 4.2, the best broadcast program will be obtained by shifting the *CacheSize* hottest pages from the fastest disk to the slowest. Such shifting is accomplished in the simulation model by setting *Offset* = *CacheSize*.[5] Given this *Offset*, we now examine the effectiveness of a cache (using the idealized $\mathcal{P}$ replacement policy) in allowing a client to tolerate *Noise* in the broadcast. Figure 8 shows the impact of increasing *Noise* on the performance of the three-disk configuration D5 as Δ is varied. In the case shown, *CacheSize* and *Offset* are both set to 500 pages. Comparing these results with the results obtained in the no caching case (see Figure 7), we see that although as expected the cache greatly improves performance in an absolute sense, surprisingly, the cache-based numbers are if anything, somewhat *more* sensitive to the degree of *Noise* than the non-caching numbers. For example, in the caching case, when Δ is greater than 2, the higher degrees of noise have multi-disk performance that is worse than the flat disk performance, whereas this crossover did not occur for similar Δ values in the non-caching case. The reason for this additional sensitivity is that when *Noise* is low and *Offset* = *CacheSize*, $\mathcal{P}$ does exactly what it should do — it caches those hot pages that have been placed on the slowest disk, and it obtains the remainder of the hottest pages from the fastest disk. However, as noise increases, $\mathcal{P}$ caches the same pages regardless of what disk they are stored on. Caching a page that is stored on the fastest disk is often not a good use of the cache, as those pages are broadcast frequently. As noise increases, $\mathcal{P}$'s cache hit rate remains the same, but its cache *misses* become more expensive, as it has to retrieve some pages from the slower disks. These expensive cache misses are the cause of $\mathcal{P}$'s sensitivity to *Noise*.

[4]It is trivial to implement $\mathcal{P}$ in the simulator, as the probability of each page is known from the client access distribution.

[5]The impact the *Offset* parameter is discussed in more detail in [Acha94].

5.4 Cost Based Replacement Algorithms

In the previous section, it was shown that while standard caching can help improve performance in a multi-disk broadcast environment, it can actually increase the client's sensitivity to *Noise*. Recall that *Noise* represents the degree to which the server broadcast deviates from what is best for a particular client. It is likely, therefore, that some type of "noise" will be present in any application in which there are multiple clients that access the broadcast disk. The $\mathcal{P}$ replacement policy was found to be sensitive to noise because it ignored the *cost of re-acquiring a page* when choosing a victim for replacement. To address this deficiency, we examine a second idealized algorithm called $\mathcal{PIX}$, that extends $\mathcal{P}$ with the notion of cost. As stated in Section 3, $\mathcal{PIX}$ always replaces the page with the lowest ratio of access probability to broadcast frequency. Thus, the cost of re-accessing a replaced page is factored into the replacement decision.

5.4.1 Experiment 4: $\mathcal{PIX}$ and Noise

Figure 9 shows the response time of the client using $\mathcal{PIX}$ for the same case that the previous experiment showed for $\mathcal{P}$ (see Figure 8). Comparing the two figures it can be seen that $\mathcal{PIX}$ is much more successful at insulating the client response time from effects of *Noise*. Of course, an increase in *Noise* still results in a degradation of performance; this is to be expected. However, unlike the case with $\mathcal{P}$, using $\mathcal{PIX}$ the performance of the client remains better than the corresponding flat disk performance for all values of *Noise* and Δ in this experiment. Under $\mathcal{PIX}$, the performance of the client for a given *Noise* value remains stable as Δ is increased beyond a certain point. In contrast, under $\mathcal{P}$, in the presence of noise, the performance of the client quickly degrades as Δ is increased beyond a value of 1 or 2. This experiment demonstrates the potential of cost-based replacement for making the broadcast disk practical for a wider range of applications.

Figure 10 shows results from the same set of experiments in a slightly different light. In this figure, the effect of increasing noise on the response time of the two algorithms for $\Delta = 3$ and $\Delta = 5$ is shown. The performance for the flat disk

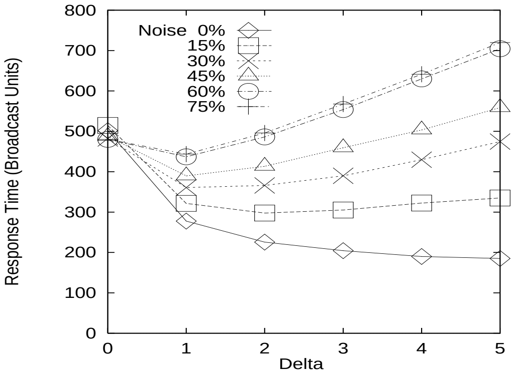

Figure 8: Noise Sensitivity - Disk D5
CacheSize = 500, Replacement Policy = $\mathcal{P}$

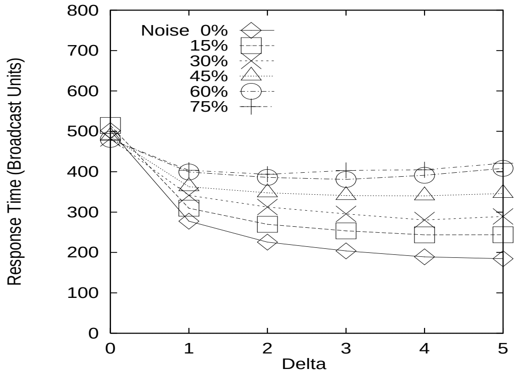

Figure 9: Noise Sensitivity - Disk D5
CacheSize = 500, Replacement Policy = $\mathcal{PIX}$

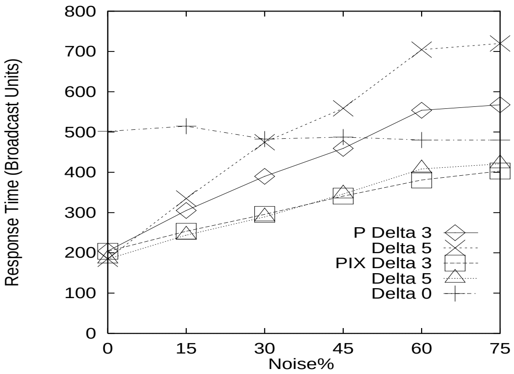

Figure 10: $\mathcal{P}$ vs. $\mathcal{PIX}$ With Varying Noise
Disk D5, *CacheSize* = 500

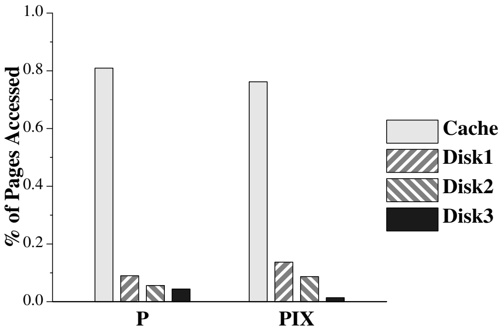

Figure 11: Access Locations for $\mathcal{P}$ vs. $\mathcal{PIX}$
Disk D5, *CacheSize* = 500, Noise = 30%, Δ = 3

($\Delta = 0$) is given as a baseline.[6] Note that $\mathcal{P}$ degrades faster than $\mathcal{PIX}$ and eventually becomes worse than the flat disk at around *Noise* = 45%. $\mathcal{PIX}$ rises gradually and manages to perform better than the flat disk within these parameters. Also, notice how $\mathcal{P}$'s performance degrades for $\Delta = 5$; unlike $\mathcal{PIX}$ it fails to adapt the cache contents with increasing differences in disk speeds.

The performance differences between the two algorithms result from the differences in the places from which they obtain their pages (as shown in Figure 11 for the case where *Noise = 30%*). It is interesting to note that $\mathcal{PIX}$ has a lower cache hit rate than $\mathcal{P}$. A lower cache hit rate does not mean lower response times in broadcast environments; the key is to reduce expected latency by caching important pages that reside on the slower disks. $\mathcal{PIX}$ gets fewer pages from the slowest disk than does $\mathcal{P}$, even though it gets more pages from the first and second disks. In this case, this tradeoff results in a net performance win.

5.5 Implementing Cost Based Policies

The previous sections have shown that multi-disk broadcast environments have special characteristics which when correctly exploited can result in significant performance gains. They also demonstrated the need for cost-based page replacement and examined a cost-based algorithm ($\mathcal{PIX}$). Unfortunately, like $\mathcal{P}$, the policy on which it is based, $\mathcal{PIX}$ is not an implementable algorithm. However, based on the insight that we gained by examining $\mathcal{P}$ and $\mathcal{PIX}$ we have designed and implemented an approximation of $\mathcal{PIX}$, which we call $\mathcal{LIX}$.

$\mathcal{LIX}$ is a modification of LRU that takes into account the broadcast frequency. LRU maintains the cache as a single linked-list of pages. When a page in the cache is accessed, it is moved to the top of the list. On a cache miss, the page at the end of the chain is chosen for replacement.

In contrast, $\mathcal{LIX}$ maintains a number of smaller chains: one corresponding to each disk of the broadcast ($\mathcal{LIX}$ reduces to LRU if the broadcast uses a single flat disk). A page always enters the chain corresponding to the disk in which it is broadcast. Like LRU, when a page is hit, it is moved to the top of *its own* chain. When a new page enters the cache, $\mathcal{LIX}$ evaluates a *lix* value (see next paragraph) only for the page at the bottom of each chain. The page with the smallest *lix* value is ejected, and the new page is inserted in the appropriate queue. Because this queue might be different than the queue from which the slot was recovered, the chains do not have fixed sizes. Rather, they dynamically shrink or grow depending on the access pattern at that time. $\mathcal{LIX}$ performs a constant number of operations per page replacement (proportional to the number of disks) which is the same order as that of LRU. Figure 12 shows an example of $\mathcal{LIX}$ for a two-disk broadcast. Pages `g` and `k` are at the bottom of each chain. Since `g` has a lower *lix* value it is chosen as the victim. The new page `z`, being picked from the second disk, joins Disk2Q. Note the relative changes in the

[6] Note that at $\Delta = 0$ (i.e., a flat disk), $\mathcal{P}$ and $\mathcal{PIX}$ are identical, as all pages are broadcast at the same frequency.

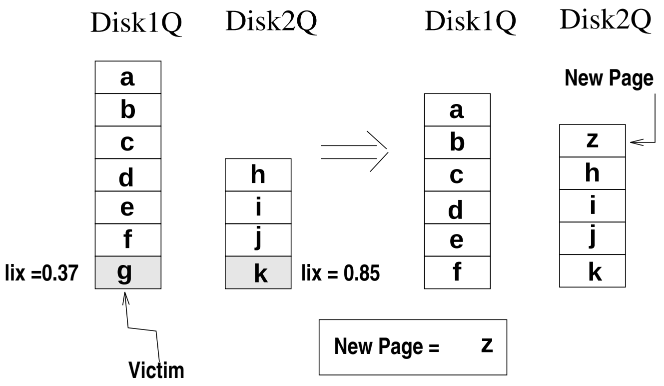

Figure 12: Page replacement in $\mathcal{LIX}$

sizes of both the queues.

In order to compute the *lix* value, the algorithm maintains two data items per cached page i : a running probability estimate (p_i) and the time of the most recent access to the page (t_i). When the page i enters a chain, p_i is initially set to zero and t_i is set to the current time. If i is hit again, the new probability estimate for i is calculated using the following formula:

$$p_i = \Lambda / (CurrentTime - t_i) + (1 - \Lambda) p_i$$

t_i is then subsequently updated to the current time. Λ is a constant used to appropriately weigh the most recent access with respect to the running probability estimate; in these experiments, it is set to 0.25. This formula is evaluated for the least recently used pages of each chain to estimate their current probability of access. This value is then divided by the frequency for the page (which is known exactly) to get the *lix* value. The page with the lowest *lix* value is ejected from the cache. $\mathcal{LIX}$ is a simple approximation of $\mathcal{PIX}$, yet in spite of this, it performs surprisingly well (as is shown below). Better approximations of $\mathcal{PIX}$, however, might be developed using some of the recently proposed improvements to LRU like 2Q[John94] or LRU-k[ONei93].

5.5.1 Experiment 5: $\mathcal{LIX}$ vs. LRU

The next set of experiments are similar to those for $\mathcal{P}$ and $\mathcal{PIX}$ and compare $\mathcal{LIX}$ and LRU. However, unlike $\mathcal{P}$, the best performance for LRU isn't at an offset equal to the cache size. Being only an approximation of $\mathcal{P}$, LRU isn't able to retain all of the hot pages that are stored on the slowest disk and thus, it performs poorly at this offset. For similar reasons, $\mathcal{LIX}$ also does not perform best at this offset. As a result, we also compared the performance of $\mathcal{LIX}$ and LRU to a modified version of $\mathcal{LIX}$ called $\mathcal{L}$. $\mathcal{L}$ behaves exactly like $\mathcal{LIX}$ except that it assumes the same value of frequency for all pages. Thus, the difference in performance between $\mathcal{L}$ and LRU indicates how much better (or worse) an approximation of probability $\mathcal{L}$ provides over LRU, and the performance difference between $\mathcal{LIX}$ and $\mathcal{L}$ shows the role that broadcast frequency plays (if any) in the performance of the caching strategies.

Figure 13 shows the performance of the three algorithms for different values of Δ. These results show the sensitivity of the algorithms to changing Δ for the same case as in Figure 10 (i.e., *Offset*=*CacheSize*=500), with *Noise* set to 30%. In this experiment, LRU performs worst and consistently degrades as Δ is increased. $\mathcal{L}$ does better at $\Delta = 1$ but then degrades. The benefits of using frequency are apparent from the difference in response time between $\mathcal{LIX}$ and $\mathcal{L}$. The response time of $\mathcal{LIX}$ is only between 25% to 50% that of $\mathcal{L}$. The solid line on the bottom of the graph shows how the ideal policy ($\mathcal{PIX}$) performs; it does better than $\mathcal{LIX}$, but only by a small margin. The factors underlying these results can be seen in Figures 14, which shows the distribution of page access locations for the results of Figure 13 when Δ is set to 3. In this case, $\mathcal{LIX}$ obtains a much smaller proportion of its pages from the slowest disk than do the other algorithms. Given that the algorithms have roughly similar cache hit rates, the differences in the distributions of access to the different disks is what drives the performance results here.

Figure 15 shows the performance of the three algorithms with varying *Noise* with $\Delta = 3$. In this case, it can be seen that $\mathcal{L}$ performs only somewhat better than LRU. The performance of $\mathcal{LIX}$ degrades with noise as expected, but it outperforms both $\mathcal{L}$ and LRU across the entire region of *Noise* values. These results demonstrate that the frequency-based heuristic of $\mathcal{LIX}$ can provide improved performance in the presence of noise.

6 Previous Work

While no previous work has addressed multilevel broadcast disks and the related cache management techniques described in this paper, several projects in mobile databases and other areas have performed related work. As stated previously, the notion of using a repetitive broadcast medium for database storage and query processing was investigated in the Datacycle project at Bellcore [Herm87, Bowe92]. Datacycle was intended to exploit high bandwidth, optical communication technology and employed custom VLSI data filters for performing associative searches and continuous queries on the broadcast data. Datacycle broadcast data using a flat disk approach and so the project did not address the multi-level disk issues that we have addressed in this paper. However, the Datacycle project did provide an optimistic form of transaction management which employed an "upstream network" that allowed clients to communicate with the host. We intend to investigate issues raised by allowing such upstream communication through low-bandwidth links as part of our ongoing work. An early effort in information broadcasting, the Boston Community Information System (BCIS) is described in [Giff90]. BCIS broadcast newspapers and information over an FM channel to clients with personal computers specially equipped with radio receivers. Like Datacycle, they too used a flat disk approach.

More recently, the mobile computing group at Rutgers has investigated techniques for indexing broadcast data [Imie94b]. The main thrust of this work has been to investigate ways to reduce power consumption at the clients in order to preserve battery life. Some of the indexing techniques described in [Imie94b] involve the interleaving of index information with data, which forms a restricted type of multilevel disk. However, this work did not investigate the notion of replicating the actual data to support non-uniform access patterns and did not investigate the impact of caching.

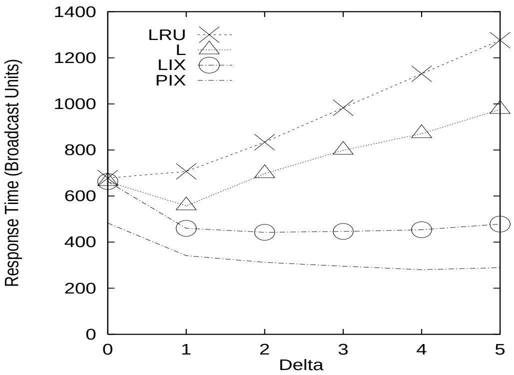

Figure 13: Sensitivity to Δ - Disk D5
CacheSize = 500, Noise = 30%

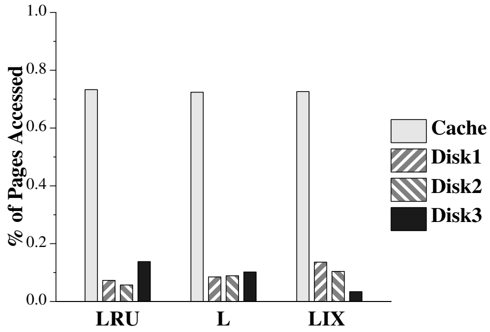

Figure 14: Page Access Locations - Disk D5
CacheSize = 500, Noise = 30%, Δ = 3

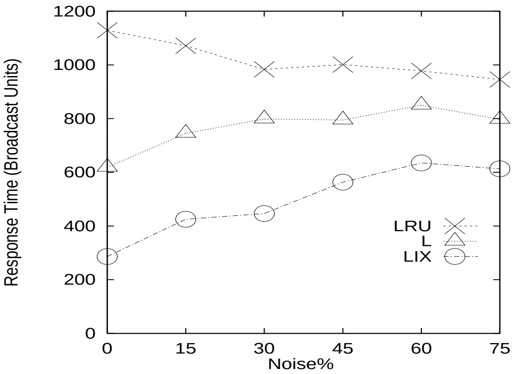

Figure 15: Noise Sensitivity - Disk D5
CacheSize = 500, Δ = 3

In our current work we have assumed a fixed broadcast program, so that indexing was not needed. However, we are currently investigating ways to integrate indexes with the multilevel disk in order to support broadcast program changes due to client population changes and updates. Caching in a mobile environment has been considered in [Barb94]. However, their model was different in that it considered volatile data and clients who could be inactive (and/or disconnected) over long periods of time. Thus, the focus of both broadcasting and caching in this work was to efficiently detect and avoid access to stale data in the cache. Very recently, another approach to broadcasting data for video on demand has been taken in [Vish94]. The technique, called pyramid broadcasting, splits an object (e.g., a video clip) into a number of segments of increasing sizes. To minimize latency the first segment is broadcast more frequently than the rest. While similar in spirit, a key difference is that the data needed by the client is known a priori once the first segment (the choice of movie) is decided upon and thus, they do not need to address the issues related to caching dealt in this paper.

The issues that arise due to our use of a broadcast medium as a multi-level device also arise in other, more traditional types of complex memory hierarchies. The need for cost-based caching and page replacement has been recognized in other domains in which there is a wide variation in the cost of obtaining data from different levels of the storage hierarchy. For example, [Anto93] describes the need for considering "cost of acquisition" for page replacement in deep-store file systems involving tertiary mass storage. This issue is also addressed for client-server database systems in which a global memory hierarchy is created by allowing clients to obtain data from other clients that have that data cached [Fran92]. In this work, server page replacement policies are modified to favor pages that are not cached at clients, as they must be obtained from disk, which is more expensive. Recently, a technique called "Disk-Directed I/O" has been proposed for High Performance Computing applications [Kotz94]. Disk-Directed I/O sends large requests to I/O devices and allows the devices to fulfill the requests in a piecemeal fashion in an order that improves the disk bandwidth. Finally, the tradeoff between replication to support access to hot data while making cold data more expensive to access has been investigated for magnetic disks [Akyu92].

7 Summary and Future Work

In this paper, we have described our design of a multilevel broadcast disk and cache management policies for this style of memory. We believe that this approach to data management is highly applicable to asymmetric network environments such as those that will naturally occur in the NII as well as many other modern data delivery systems. We have demonstrated that in designing such disks, the broadcast program and the caching policy must be considered together.

It has been shown that there are cases in which the performance of both two and three level disks can outperform a flat broadcast even when there is no caching. We have argued that our scheme for interleaving the data is desirable because it provides a uniform expected latency.

We have further shown that introducing a cache can provide an advantage by smoothing out disagreement between the broadcast and the client access patterns. The cache gives the clients a way to hoard their hottest pages regardless of how frequently they are broadcast. However, doing page replacement solely on probability of access can actually increase a client's sensitivity to the server's broadcast.

We then introduced a caching policy that also took into account the broadcast frequency during replacement. We showed that this not only improves client performance but also shields it from vagaries of the server broadcast. This is because the clients can cache items that are relatively hot and reside on a slow disk and thus, avoid paying high cache miss penalties.

Finally, we demonstrated a straightforward implementation technique that approximates our ideal cost-based caching scheme. This technique is a modification of LRU which accounts for the differences in broadcast frequency of the data.

We believe that this study while interesting and useful in its own right, is just the tip of the iceberg. There are many other opportunities that can be exploited in future work. Here, we have only considered the static read-only case. How would our results have to change if we allowed the broadcast data to change from cycle to cycle? What kinds of changes would be allowed in order to keep the scheme manageable, and what kinds of indexing would be needed to allow the client to make intelligent decisions about the cost of retrieving a data item from the broadcast?

We are currently investigating how prefetching could be introduced into the present scheme. The client cache manager would use the broadcast as a way to opportunistically increase the temperature of its cache. We are exploring new cache management metrics for deciding when to prefetch a page.

We would also like to provide more guidance to a user who wants to configure a broadcast. We have experimental results to show that good things can happen, but given a workload, we would like to have concrete design principles for deciding how many disks to use, what the best relative spinning speeds should be, and how to segment the client access range across these disks. We are pursuing an analytic model to address this.

Finally, once the basic design parameters for broadcast disks of this kind are well-understood, work is needed to develop query processing strategies that would exploit this type of media.

Acknowledgements

The authors would like to thank M. Ranganathan for providing them with a number of important insights into the properties of broadcast programs. Franklin's research was supported in part by a grant from the University of Maryland General Research Board, grant number IRI-9409575 from the NSF and by a gift from Intel Corporation. Acharya and Zdonik were supported in part by ONR grant number N00014-91-J-4085 under ARPA order number 8220 and by a gift from Intel Corporation.

References

[Acha94] S. Acharya, R. Alonso, M. Franklin, S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communications Environments", Tech. Report CS-94-43, Brown Univ.; Tech. Report CS-TR-3369, Univ. of Maryland, Oct. 1994.

[Akyu92] S. Akyurek, K. Salem, "Placing Replicated Data to Reduce Seek Delays" *Proc. USENIX File System Conf.*, May 1992.

[Anto93] C. Antonelli, P. Honeyman, "Integrating Mass Storage and File Systems", *Proc. 12th IEEE Symp on Mass Storage Sys.*, 1993.

[Barb94] D. Barbara, T. Imielinski, "Sleepers and Workaholics: Caching Strategies in Mobile Environments",*Proc. ACM SIGMOD Conf.*, May, 1993.

[Bowe92] T. Bowen, et al. "The Datacycle Architecture" *CACM* 35,(12), Dec., 1992.

[Dan90] A. Dan, D. M. Dias, P. Yu, "The Effect of Skewed Access on Buffer Hits and Data Contention in a Data Sharing Environment", *Proc. 16th VLDB Conf.*, Aug., 1990.

[Fran92] M. Franklin, M. Carey, M. Livny, "Global Memory Management in Client-Server DBMS Architectures", *Proc. 18th VLDB Conf.*, Aug., 1992.

[Giff90] D. Gifford, "Polychannel Systems for Mass Digital Communications",*CACM*, 33(2), Feb., 1990.

[Gray94] J. Gray, et al., "Quickly Generating Billion-Record Synthetic Databases", *Proc. ACM SIGMOD Conf.*, May, 1994.

[Herm87] G. Herman, G. Gopal, K. Lee, A. Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems", *Proc. ACM SIGMOD Conf.*, May, 1987.

[Imie94a] T. Imielinski, B. Badrinath, "Mobile Wireless Computing: Challenges in Data Management", *CACM*, 37(10), Oct., 1994.

[Imie94b] T. Imielinski, S. Viswanathan, B. Badrinath, "Energy Efficient Indexing on Air" *Proc. ACM SIGMOD Conf.*, May, 1994.

[John94] T. Johnson, D. Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm", *Proc. 20th VLDB Conf.*, Sept., 1994.

[Katz94] R. Katz, "Adaption and Mobility in Wireless Information Systems", *IEEE Personal Comm.*, 1st Quarter, 1994.

[Knut81] D. Knuth, "The Art of Computer Programming, Vol II", Addison Wesley, 1981.

[Kotz94] D. Kotz, "Disk-directed I/O for MIMD Multiprocessors", *1st Symposium on OS Design and Implementation*, USENIX, Nov., 1994.

[ONei93] E. J. O'Neil, P. E. O'Neil, G. Weikum, "The LRU-k Page Replacement Algorithm for Database Disk Buffering", *Proc. ACM SIGMOD Conf.*, May, 1993.

[Schw86] H. D. Schwetman, "CSIM: A C-based process oriented simulation language",*Proc. 1986 Winter Simulation Conf.*, 1986.

[Vish94] S. Vishwanath, T. Imielinski, "Pyramid Broadcasting for Video on Demand Service", Rutgers Univ. Tech. Report DCS TR-311, 1994.

[Zdon94] S. Zdonik, M. Franklin, R. Alonso, S. Acharya, "Are 'Disks in the Air' Just Pie in the Sky?", *IEEE Wkshp on Mobile Comp. Sys. and Applications*, Santa Cruz, CA, Dec., 1994.