Dissemination-based Data Delivery Using Broadcast Disks

Swarup Acharya* Michael Franklin† Stanley Zdonik‡

Abstract

Mobile computers and wireless networks are emerging technologies which promise to make ubiquitous computing a reality. One challenge that must be met in order to truly realize this potential is that of providing mobile clients with ubiquitous access to data. Mobile clients may often be disconnected from stationary server machines or may have only a low-bandwidth channel for sending messages to servers. Such an environment raises difficulties for supporting data-intensive applications for three reasons: 1) the inability to predict, with 100% accuracy, the future data needs of many applications, 2) limits on storage capacities of mobile machines, and 3) the need to provide clients with new or updated data values. One (and perhaps the only) way to address these challenges is to provide stationary server machines with a relatively high-bandwidth channel over which to broadcast data to a client population in anticipation of the need for that data at the clients. Such a system can be said to be asymmetric due to the disparity in the transmission capacities of clients and servers. We have proposed a mechanism called *Broadcast Disks* to provide database access in this environment as well as in other asymmetric systems such as cable and direct broadcast satellite television networks and information distribution services. The Broadcast Disk approach enables the creation of an arbitrarily fine-grained memory hierarchy on the broadcast medium. This hierarchy, combined with the inversion of the traditional relationship between clients and servers that occurs in a broadcast-based system raises fundamental new issues for client cache management and data prefetching. In this paper we present a brief overview of asymmetric environments and describe our approaches to broadcast disk organization, client cache management, and prefetching.

1 Introduction

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

*Dept. of Computer Science, Brown University, Providence, RI 02912 (sa@cs.brown.edu)

†Dept. of Computer Science, University of Maryland, College Park, MD 20742, (franklin@cs.umd.edu)

‡Dept. of Computer Science, Brown University, Providence, RI 02912 (sbz@cs.brown.edu)

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.

In the Broadcast Disks project [Zdon94, Acha95, Acha96], we are investigating the use of data broadcast in order to exploit the server's advantage in communication capacity in asymmetric environments. In this approach, a server *continuously and repeatedly* broadcasts data to a client community. In effect, the broadcast channel becomes a "disk" from which clients can retrieve data as it goes by. Data broadcast has been addressed previously by other researchers in both wired and wireless networks (e.g., [Amma85, Herm87, Giff90, Imie94b]). Our work differs from this previous work in two important ways. First, we superimpose *multiple disks* of different sizes and speeds on the broadcast medium; data items that are more likely to be of interest to a larger part of the client community can be broadcast more frequently than others. This in effect creates an *arbitrarily fine-grained memory hierarchy*, as the expected delay in obtaining an item depends upon how often that item is broadcast. Second, a major focus of our work has been on the exploitation of client storage resources for caching data in order to lessen a client's dependency on the server's choice of broadcast priority and for prefetching from the broadcast in order to reduce the expected delay for accessing data.

1.1 Applications

Much of our work to date has been driven by wireless applications. One of the anticipated problems in a wireless environment is that of *elective* disconnection [Kist92, Imie94a]. Mobile users may be disconnected from the server due to the large (monetary) cost or the unavailability of an uplink connection. In the absence of data broadcast, disconnected users must rely solely on the data cached locally (possibly including data stored on removable memory devices). However, this data can be insufficient for several reasons. First, in many situations, it may not be possible to predict with complete accuracy all of the data that will be accessed during the disconnected period. Secondly, data items that are downloaded prior to disconnection can become stale as the result of updates by other users or at the server itself (e.g., in a stock-quotation application). In addition, new data items of importance to a client may be created while that client is not physically connected to a server.

While elective disconnection in a wireless environment is an important issue, such disconnection can simply be viewed as an extreme case of asymmetric communication. Therefore, many of the data management issues that arise in the wireless mobile environment can also arise in the other asymmetric environments listed above, even those involving wired networks. For example, in the case of cable TV, the number of available uplink channels is limited, and thus, we would like to avoid using them when possible. The broadcast disk is an effective way to get data to clients in many of these situations.

One of the major advantages of the broadcast disk paradigm is its scalability; the performance of the system does not depend on how many users are listening. The broadcast disk, therefore, is an ideal medium for disseminating public information such as weather and financial reports. Among the many potential applications of wireless broadcast disks in the near future, one that is ideally suited is the Advanced Traffic Information System(ATIS) [Shek94]. An ATIS involves several host machines that contain current information on traffic patterns in a metropolitan area. Drivers carry a small, special-purpose computer in their car to assist them in tasks such as planning routes. The broadcast could be used for sending static information such as street topology as well as dynamic information like the locations of certain traffic problems.

The client population for a moderately sized metropolitan area might be 200,000 cars. Moreover, these cars will likely not be connected to the server most of the time. Since the data of interest (e.g., accidents, traffic jams etc.) is inherently volatile, these cannot be pre-loaded on the client machine. Given the two options available for transferring data: 1) having individual clients query the server or 2) having the server broadcast the data, the latter has several advantages. A system based on querying suffers from scalability and problems of disconnection. Furthermore, given the similar nature of the queries that will be asked by the motorists, broadcasting can amortize much of the overhead of sending messages across multiple clients. If the broadcast bandwidth is limited, then only dynamic information need be sent. Relatively static information, such as road maps, could be supplied on CD-ROM.

1.2 Broadcast Disks

The multiple-disk approach to data broadcast that is advocated by the Broadcast Disk technique poses many challenging research problems. There are two major sets of issues to be addressed in developing the broadcast disks approach — client side and server side. The main challenge from the server's perspective is the organization of the data in the broadcast. The serial nature of the medium precludes random access; therefore the distribution of data on the broadcast is a crucial factor in performance. Furthermore, because bandwidth is essentially a fixed resource, increasing the frequency of broadcast for some items must necessarily reduce the frequency of broadcast for other items. The problems that arise on the clients' side are related to page replacement and prefetching. Our previous experiments have shown the inadequacy of traditional caching techniques in broadcast environments. Because some data items are broadcast far less frequently than others, cache management decisions must include the cost of re-access in addition to probability of access. As would be expected, the varying costs of obtaining items from the broadcast also impacts the way in which prefetching must be done.

In this paper we identify four specific design issues: 1) structuring the broadcast, 2) managing client data caches in the presence of widely varying expected access times for data items, 3) techniques for prefetching data from the broadcast, and 4) dealing with changes to the data being broadcast and in the nature of the clients' access needs. The bulk of our work so far has been performed in the context of a restricted broadcast environment in which data is accessed in a read-only manner, the client population and its access demands are static, and clients do not exploit an upstream communications link. Even in this constrained setting, we have obtained some very significant results. The remainder of the paper highlights these issues, presents an overview of some of our approaches towards addressing them, and outlines important areas for future work.

2 Server Broadcast Programs

2.1 Properties of Broadcast Programs

In a broadcast-based information system, a data 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 required items and broadcast the resulting set cyclicly. Such a *flat* broadcast is depicted in Figure 1. A flat broadcast can be thought of as a single additional layer in a client's memory hierarchy, residing below the local storage resources of the client. That is, when an application running on a client needs a data item, it first attempts to retrieve that item from the local memory or disk and if the desired item is not found, then the client monitors the broadcast and waits for the item to arrive.[1]

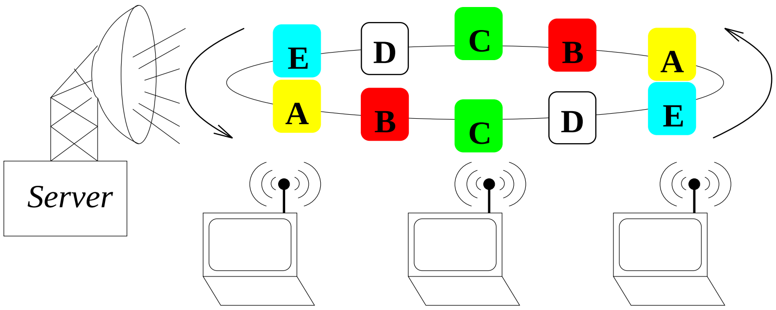

Figure 1: A Flat Broadcast Program

With a 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 [Herm87, Bowe92] and the data broadcasting work at Rutgers [Imie94b]. In the Broadcast Disks approach, we advocate an alternative broadcast organization. Rather than treating the broadcast medium as a single, flat disk, the broadcast can be structured as multiple disks of varying sizes, each spinning at a different rate. By making fast disks small and slower disks increasingly larger, this approach generalizes to an arbitrarily fine-grained memory hierarchy. Intuitively, the allocation of data items to the various disks would therefore involve tradeoffs analogous to those found in the construction of more traditional memory hierarchies, such as "The Five Minute Rule" [Gray87]. Thus, the items in highest demand (*hot* items) would be placed on the fastest spinning disks; the *colder* items would be placed on the bigger, slower disks. Consequently, if a client faults on a *hot* data item, the expected delay can be considerably shorter in comparison to a flat disk. However, the delay in accessing colder items is longer than with a flat disk, introducing an interesting tradeoff in broadcast program organization.

Conceptually, the generation of 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. Such a random broadcast, however, will not be optimal in terms of minimizing expected delay due to the variance in the inter-arrival times.

A simple example demonstrating the tradeoffs is shown in Figure 2. The figure shows three

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

different broadcast programs for a data set containing three equal-length items (e.g., pages).[2] 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 potentially 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.

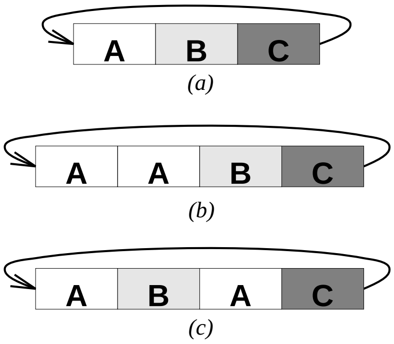

Figure 2: Three Example Broadcast Programs

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*

[2] For simplicity, we assume throughout this paper that the unit of broadcast is a "page", i.e., all broadcast items are of a uniform, fixed length.

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 related to the so-called Bus Stop Paradox. If the inter-arrival 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 [Acha96] (prefetching is discussed in Section 4). Second, the randomness of broadcast disallows the use of "sleeping" to reduce power consumption (as in [Imie94b]). Third, as discussed in Section 3, the grouping of data items into a small number of classes based on broadcast frequency can be exploited in the implementation of client caching policies. Finally, there is no notion of "period" for a random broadcast. Periodicity may be important for providing correct semantics for updates (e.g., as was done in Datacycle) and for introducing changes to the contents 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).
- There should be a small number of different broadcast frequencies.
- Subject to the above constraints, as much of the available broadcast bandwidth should be used as possible.

2.2 Generating a Multi-Disk Broadcast

In [Acha95] we describe a broadcast disk model that can be used to achieve the above goals. The 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, and the relative frequency of broadcast (*rel_freq(i)*) can be specified. These parameters 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.

Given a list of pages ordered by their expected access probabilities[3] and a disk shape specification as described above, the pages can be assigned to disks and the disks can be interleaved to achieve the desired broadcast schedule. 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 D1 are to be broadcast twice as frequently as pages in D2, and four times as frequently as pages in D3. Therefore, *rel_freq(1)* = 4, *rel_freq(2)* = 2, and *rel_freq(3)* = 1. These disks are split into a number of "chunks" based on the Least Common Multiple (LCM) of the relative frequencies of broadcast for the various disks. In this case, the LCM is 4, so D3, the slow disk is split into 4 chunks, D2, which spins at twice the speed of D3 is split into 2 chunks, and D1, which spins at four times the speed of D3 is kept as a single chunk. 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, and has a period of 16 pages. This broadcast creates a three-level memory hierarchy in which D1 is the smallest and fastest level and D3 is the largest and slowest level. Thus, the broadcast corresponds to the traditional notion of a memory hierarchy.

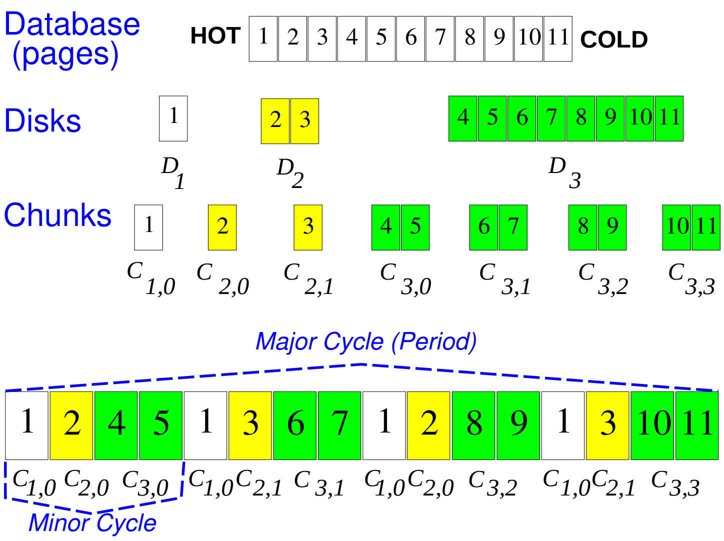

Figure 3: Deriving a Server Broadcast Program

This approach 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. 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

[3] We address the sensitivity of the Broadcast Disk performance to the accuracy of these expected access probabilities in Section 3.

broadcasts of pages that are requested in a demand-driven fashion by clients using an uplink channel. Furthermore, if the number of disks is kept small (on the order of 2 to 5) and the number of pages to be broadcast is substantially larger, then unused slots (if any) will be only a small fraction of the total number of slots. In addition, the relative frequencies of the disks can be adjusted slightly to reduce the number of unused slots, if necessary.

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 approach 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 an interesting optimization problem, and is one focus of our on-going work.

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, caching-based systems (e.g., [Arch86, Howa88, Wilk90, Care91, Wang91] etc.), clients cache their *hottest* data (i.e., the items that they are most likely to access in the future). In the Broadcast Disk 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 due to the fact that in a multi-disk broadcast, pages are *not* equidistant from the client.

In general, 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, clients in a Broadcast Disk system 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 considered relatively unimportant, and hence, it will be broadcast on a slow disk. To avoid long waits for that 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 approximates). A simple cost-based 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 (i.e., half as often) at the client, but is broadcast only 0.1% of the time (i.e, one tenth as often). 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. This behavior, as shown in our detailed studies [Acha95], demonstrates an interesting tradeoff. Because $\mathcal{PIX}$ may sometimes favor a colder page over a hotter one, it typically has a *lower* cache hit rate than a replacement policy based purely on probability of access (e.g., idealized LRU). Despite this fact, $\mathcal{PIX}$ performs better than such a policy, because it lowers the penalty paid on a cache miss. It is interesting to note that this result has implications beyond the broadcast disk approach; it is applicable to other complex memory hierarchies such as those using tertiary storage.

While $\mathcal{PIX}$ is an optimal policy under certain conditions, it is not a practical policy 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 developed an algorithm that adds frequency of broadcast to an LRU-style policy. This new policy is called $\mathcal{LIX}$ [Acha95]. Briefly, $\mathcal{LIX}$ maintains a separate list of cache-resident pages for each logical disk. When a page is brought into the cache, it is placed in the list that corresponds to the logical disk on which it is broadcast (thus, these lists can dynamically change size). Each list is ordered based on an approximation of the access probability for each of the pages (similar to an LRU stack). This estimate, which we call L, is based on a running average of inter-access times for the page; hysteresis is used in order to favor more recent history in this metric. At page replacement time, only the page with the lowest L value for each disk need be examined. For each such page, a $\mathcal{LIX}$ value is computed by dividing L by the page broadcast frequency X, which is constant for each disk. The page with the lowest $\mathcal{LIX}$ value is then chosen as the page replacement victim.

In order to investigate the performance of the different cache replacement policies described above, we have implemented a simulation environment that models a single server, which continually broadcasts pages according to a specified broadcast program, and a single client which makes random page requests according to a specified distribution. The experiments and results are presented in [Acha95]; a detailed description of that work is beyond the scope of this overview. Here we show a sample of those results that demonstrate some of the major performance tradeoffs discussed above. Figures 4 and 5 show response time results for a broadcast with three logical disks (containing 300,1200, and 3500 pages, respectively) and a client cache size of 500 pages. In these experiments the database contains 5000 pages, of which the client accesses 1000 using a Zipf distribution [Gray94], which has a high degree of locality (i.e., access skew). The figures show the average response time for a random page request by a client. The response times are measured in "broadcast units" (i.e., the amount of time it takes to broadcast one page) so the results are independent of a specific network bandwidth.

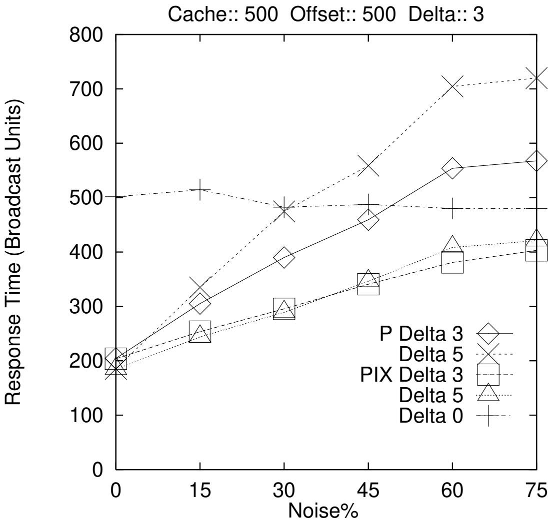

Figure 4: $\mathcal{P}$ vs. $\mathcal{PIX}$ With Varying Noise

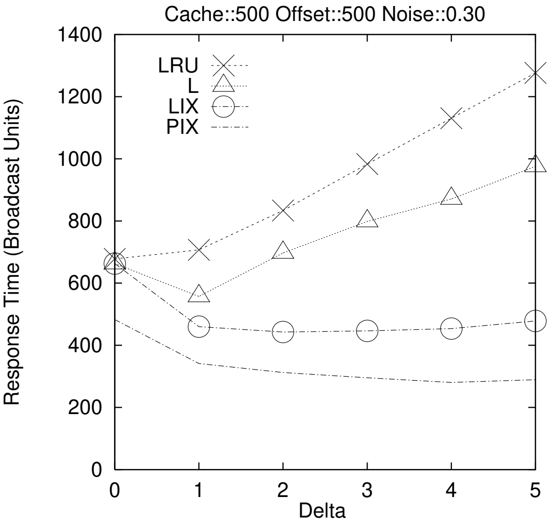

Figure 5: $\mathcal{LIX}$ vs. Others With Varying Δ

The results shown demonstrate the sensitivity of the various replacement policies to two parameters that are used in our simulations: *Noise* and Δ. The Noise parameter is intended to model the impact of a diverse client population on the performance of the single client whose performance is being measured. Noise represents the degree to which the server's broadcast program differs from what would be best for the client's access pattern — as Noise is increased, the server is less able to tailor its broadcast program to the needs of our specific client. More specifically, the Noise value is the probability that a page is placed on the wrong disk for this client (e.g., a page that should be placed on a fast disk is instead placed on a slow disk). Thus, at a Noise value of 0%, the server broadcast is perfectly matched to the client's access pattern. As Noise is increased, more pages are moved from their preferred location representing the fact that the server's broadcast is less well matched to this particular client's needs. The second simulation parameter, called Δ, is a measure of how fast the disks are spinning in relation to each other — D2 rotates $(\Delta + 1)$ times faster than D3, and D1 rotates $(2\Delta + 1)$ times faster than D3. For example, at $\Delta = 0$, all disks spin at the same speed (in effect, the disk is flat), while at $\Delta = 5$, D1 rotates 11 times faster than D3.[4]

Figure 4 demonstrates the performance of $\mathcal{PIX}$ compared to $\mathcal{P}$, which is a replacement policy based purely on the probability of access (i.e., idealized LRU). The Noise parameter is increased along the x-axis, and results are shown for three different Δ values. At $\Delta = 0$ (i.e., a flat broadcast) $\mathcal{P}$ and $\mathcal{PIX}$ have identical performance as all pages have the same broadcast frequency. Likewise, at a Noise value of 0%, $\mathcal{P}$ and $\mathcal{PIX}$ have similar performance because the frequency of broadcast for each item is matched to its probability of access. The differences between the two approaches appear when the disk speed skew (Δ) and the Noise are increased. As can be seen in the figure, $\mathcal{P}$, the policy that ignores broadcast frequency suffers a severe degradation in performance as these two parameters are increased. In contrast, $\mathcal{PIX}$ is much more robust in the presence of noise, and its performance degradation is only slightly increased by the addition of disk skew ($\Delta = 5$ vs. 3) in this case. This is because $\mathcal{PIX}$ is able to keep hot pages that have been placed on the slow disks in cache, while $\mathcal{P}$ is insensitive to the broadcast frequency of pages.

Figure 5 shows the performance of $\mathcal{LIX}$ compared to $\mathcal{PIX}$, which is an idealized policy, LRU, which is the traditional approximation to $\mathcal{P}$, and a policy called $\mathcal{L}$, which is an implementable approximation to $\mathcal{P}$, which uses the same access probability estimate as $\mathcal{LIX}$. $\mathcal{L}$ is used to measure the impact of the improved probability estimate of $\mathcal{LIX}$ in the absence of broadcast frequency information. In this figure, the disk speed skew (Δ) is varied along the x-axis starting from a flat disk. The Noise parameter in this case is fixed at 30%. As expected, $\mathcal{PIX}$, which has perfect knowledge of access probabilities, performs better than the three implementable policies throughout the entire range of Δ values. It is used in this experiment as a baseline. With a flat disk, the three implementable policies have similar performance, but these quickly diverge as Δ is increased. $\mathcal{LIX}$ has consistently better performance than the others here. In fact, its behavior approximates that of $\mathcal{PIX}$, with a penalty due to the inaccuracy of its probability estimation. In contrast, the other policies, which ignore broadcast frequency, are shown to be very sensitive to the disk speed skew in this case. These results, and others in our study have shown that $\mathcal{LIX}$ is a reasonable,

[4] It should be noted that Δ is used to restrict the shapes of disks in order to decrease the simulation parameter space, it is *not* a part of the Broadcast Disks model as described in Section 2.

implementable cost-based caching policy for the broadcast disk environment.

4 Prefetching

The previous section discussed the management of client caches when pages are brought into the cache in a demand-driven fashion. An alternative approach to obtaining pages from the broadcast is to have clients *prefetch* pages. The Broadcast Disk environment is particularly conducive to prefetching because pages are continuously presented to clients via the broadcast, making it possible for a client to bring some of these pages into its cache in anticipation of future accesses. The dissemination-oriented nature of this environment changes the tradeoffs that are commonly associated with prefetching in more traditional environments, such as database systems and file systems. In particular, prefetching from disk in a traditional system is a risky proposition, because prefetching places additional load on shared resources (e.g., disks and the network), and can therefore negatively impact the performance of the entire client population. In contrast, when prefetching from a Broadcast Disk, only the local client resources are impacted. Thus, the risks involved in prefetch are significantly lower here than in a more traditional system.

4.1 Tag Team Caching

For Broadcast Disks, the goal of prefetching is to improve the response time of clients that access data from the broadcast. Response time improvements for data requests can typically be obtained in two ways: 1) by improving the client cache hit rate, and 2) by reducing the cost of a cache miss. As stated in the previous section, the $\mathcal{PIX}$ algorithm and its approximation $\mathcal{LIX}$, have been shown to achieve an effective balance between these two issues. They both accept a slightly lower client hit rate in order to achieve a reduction in the number of pages that must be obtained from the slower disks in a multi-disk broadcast.

$\mathcal{PIX}$ and $\mathcal{LIX}$ however, are designed for a strictly demand-driven environment in which a page is brought from the broadcast into a client cache only as the result of a request for that page. The introduction of prefetching provides an additional way for pages to be brought into the cache, the potential benefits of which are illustrated by the following simple example. Consider a client that is interested in accessing two pages (x and y) with equal probability (i.e., $p_x = p_y = 0.5$), and has only a single cache slot available to use for storing one or the other of these pages. Assume that (as shown in Figure 6), the server places these two pages 180 degrees apart on the broadcast (for simplicity, a single, flat disk is used).

Under a demand-driven strategy, the client would cache one of the pages, say x, as the result of a request for that page. Subsequent accesses to x can be satisfied from the cached copy, and thus have a delay of zero. If however, the client needs page y, it waits for y to come by on the broadcast, and then replaces x with y in the cache. Page y then remains in the cache until x is

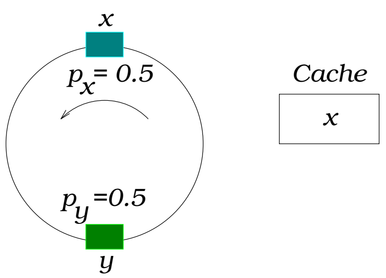

Figure 6: Tag-team Caching

faulted in. With this strategy, the expected delay on a cache miss is one half of a rotation of the disk (because the request will be made at a random time). The expected delay over all accesses can be computed as the sum of the expected costs for all pages. The expected cost of accessing a page i is its probability of access(P_i) times its average cost of access(C_i). The expected cost of access for this *demand-driven* strategy is therefore:

$$P_x C_x + P_y C_y = 0 + 0.5 * 0.5 = 0.25$$

that is, one quarter of a disk revolution.

In contrast, consider a simple prefetching strategy that fetches page x into the cache when it arrives on the broadcast and holds it until page y arrives on the broadcast. At that point, the client caches page y and drops x. When x is broadcast again, y is dropped and x is cached. We call this strategy *tag-team caching* because pages x and y continually replace each other in the cache. The expected cost of the tag-team strategy for this example is:

$$\sum_{i \in \{x,y\}} P_i(Prob_{i\ in\ cache} C_{i\ in\ cache} + Prob_{i\ not\ in\ cache} C_{i\ not\ in\ cache})$$

$$= 0.5(0 + 0.5 * 0.25) + 0.5(0 + 0.5 * 0.25) = 0.125$$

Thus, the average cost of access using tag-team is one eighth of a disk revolution — one *half* the expected cost of the demand-driven strategy.

As can be seen in the equations above, the performance gain of tag-team comes not from an improved hit rate (the hit rate is 50% for both strategies), but rather because the cost of a miss using tag-team is half of the cost of a miss under the demand-driven strategy. With the demand-driven strategy, a miss can occur at any point in the broadcast. In contrast, using tag-team, misses can only occur during half of the broadcast. For example, a miss on a reference to page x can only occur during the part of the broadcast between the arrival (and prefetching) of page y and the subsequent arrival of page x. Thus, tag-team reduces the cost of cache misses by ensuring that a page is cached during the portion of the broadcast during which the wait for that page would be the longest.

This simple tag-team example can be generalized to multiple numbers of pages with differing access probabilities and different sized caches. Briefly, the problem of assigning pages to "teams" and placing them on the broadcast can be formulated as a bin-packing problem with as many bins as there are cache slots and with the height of the bins as the number of page broadcasts in a complete cycle of the disk. The height of the items to be packed is the amount of time that each item should spend in the cache.

4.2 A Simple Prefetching Heuristic

In our work to date [Acha96], we have focused on a much simpler prefetching heuristic called $\mathcal{PT}$, which is shown to have similar behavior to the tag-team approach without the need for extensive analysis as required by the generalized tag-team. $\mathcal{PT}$ is a dynamic policy that performs a calculation for each page that arrives on the broadcast in order to decide whether that page is more valuable than some other page that is currently in the cache. $\mathcal{PT}$ uses a measure (called the `pt` value) which is the product of the probability (P) of access for a page and the amount of time (T) that will elapse before that page again appears on the broadcast. If the `pt` value of the current page being broadcast is greater than the page with the lowest `pt` value in the cache, then the latter page is chosen as the replacement victim, and the page being broadcast is brought into the cache.

$\mathcal{PT}$ is similar to the $\mathcal{PIX}$ metric described earlier in that it includes both probability of access and a notion of the cost of obtaining a page from the broadcast. It differs however, in that `pt` values are dynamic — they change with every "tick" of the broadcast. If a static measure such as $\mathcal{PIX}$ were used for prefetching, the cache would quickly fill with the highest $\mathcal{PIX}$-valued pages and then stabilize. In contrast, using $\mathcal{PT}$, pages are continually cycled through the cache in a manner similar to the tag team example described in the previous section.

The `pt` value of a page is constantly changing because the time parameter of the metric is constantly changing. When a page is broadcast, `t` is highest since ignoring that page at that point will put the client at a maximal distance from the page. In other words, the moment that a page passes by is the moment at which there will be the longest wait to get it again. From that point on, the time parameter steadily decreases with each clock tick, thus creating a sawtooth function of time for $\mathcal{PT}$. The differences between the metrics used by $\mathcal{PT}$ and $\mathcal{PIX}$, can be seen in the following example. Consider three pages, A, B, and C. As shown in Table 2, page A is accessed twice as often as pages B or C; pages A and B are broadcast twice per period and page C is broadcast only once per period. $\mathcal{PIX}$ would value pages A and C equally, and would value them twice as much as page B.

In contrast, the `pt` metric used by $\mathcal{PT}$ is a sawtooth function that changes over time . Figure 7 traces the saw tooth effect for the three pages. Assume that page A is broadcast at time units 5, 15, 25,...; page B is broadcast at time units 2, 12, 22, ...; and page C is broadcast at time units 4,

Page	**Access Probability**	**Broadcast Frequency (per Period)**	$\mathcal{PIX}$ **Value**
A	P	2	P/2
B	P/2	2	P/4
C	P/2	1	P/2

Table 2: Access and Frequency Values for the `pt` Example

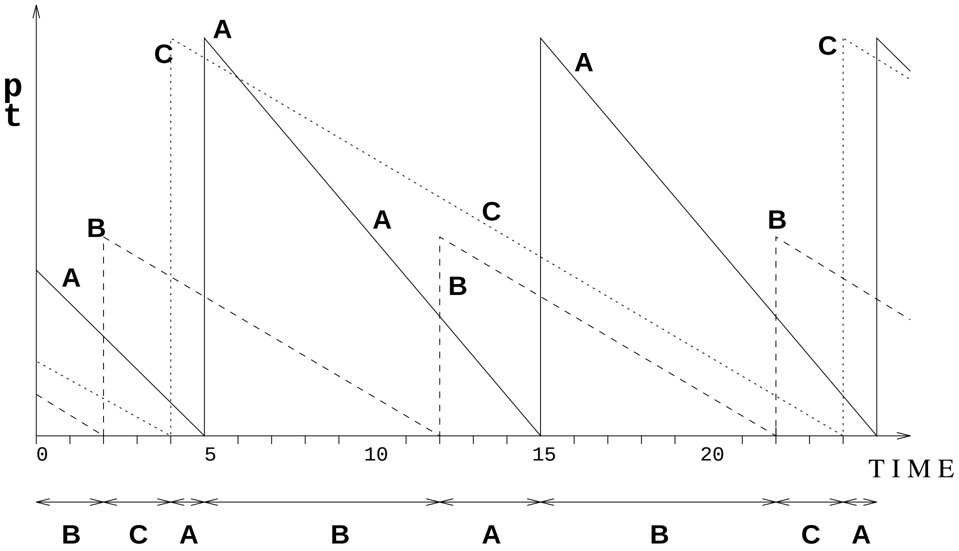

Figure 7: `pt` Value vs. Time

24, 44, ...;. As can be seen in the figure, the `pt` value for a page peaks at the instant the page is broadcast and decreases linearly until the subsequent re-broadcast of the page. The line along the bottom of the figure shows the potential replacement victim at each unit of time. Note that unlike $\mathcal{PIX}$, in which page B is always the least-valued page of the three, there are time intervals when $\mathcal{PT}$ would choose C or even A as the page replacement victim.

Because of this behavior, $\mathcal{PT}$ tends to drop high probability items in the period just before they re-appear on the broadcast. Items are stickiest just after they come into the cache since this is when their time parameter is highest. $\mathcal{PT}$ is in effect adjusting the allocation of cache slots to pages so that (subject to cache size) they are likely to be in the cache during those portions of the broadcast cycle in which they would be most expensive to obtain from the broadcast. This effect is similar to what the tag team caching approach aims to achieve. In fact, in our simulation studies of $\mathcal{PT}$ [Acha96], we observed that $\mathcal{PT}$ was able to achieve the performance of the tag team example described in the previous section.

Figure 8 shows the results of running $\mathcal{PIX}$ and $\mathcal{PT}$ on a *flat* disk containing 3000 pages, of which the client access 1000 using a Zipf distribution. The cache size is increased from 1 page (i.e., no prefetching is possible) to 1000 pages (i.e., no prefetching is necessary because all accessed pages are cache-resident). In the figure, it can be seen that at a cache size of 500 pages, the response time of $\mathcal{PT}$ is half of that of $\mathcal{PIX}$. This is what is predicted by the tag team example.

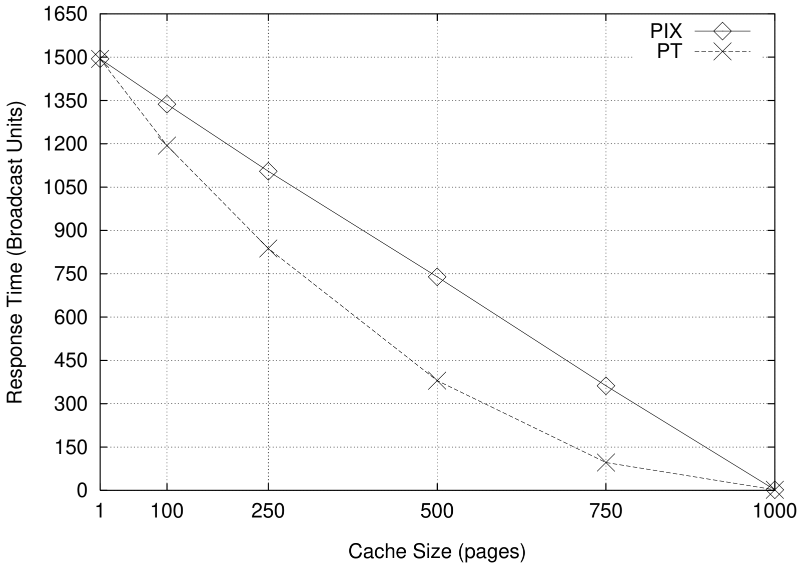

Figure 8: $\mathcal{PT}$ vs. $\mathcal{PIX}$ for a Flat Broadcast

It should be noted that any time-based prefetch scheme is potentially very expensive to implement, since it could require computing the metric for every item in the cache at every clock tick. Thus, similarly to the approach taken towards client caching described in the previous section, we have used $\mathcal{PT}$ as an idealized policy to guide our study and to develop intuition, and have then developed implementable approximations to it. We have designed and studied one such algorithm, called $\mathcal{APT}$ (for Approximate $\mathcal{PT}$), which, similarly to $\mathcal{LIX}$, uses multiple queues to restrict the number of pages that must be examined. In contrast to $\mathcal{LIX}$, however, the queues in $\mathcal{APT}$ are split based on different *access probability* regions, rather than based on broadcast frequency. The $\mathcal{PT}$ and $\mathcal{APT}$ approaches are described and analyzed in detail in [Acha96].

5 The Read/Write Case

An important extension of the work described in the previous sections is to address cases in which the broadcast can change. The ability to do dynamic broadcasts is essential for applications like ATIS, stock market monitors etc. which deal with inherently volatile data. There are at least three different changes that must be handled: 1) changes to the value of objects being broadcast, 2) reorganization of the broadcast, either by changing the allocation of pages to disks or changing the shape of the broadcast itself, and 3) changes to the contents of the broadcast in terms of bringing in new pages and removing existing pages.

Values in the broadcast can change when some connected client commits an update (if an uplink channel is used) or when the broadcast application itself generates new data (say, a stock market monitor). This change may be of interest to other clients, including disconnected clients, if they have previously cached this value. The broadcast disk can be used as a medium for communicating the change in the form of a new value or an invalidation of the old value.

Furthermore, the actual content of the broadcast disk might change. Suppose that we are broadcasting stock prices. A given stock price that is not in the broadcast may become interesting when its price changes; it should be moved onto the broadcast, perhaps displacing something else. Alternatively, if the stock price suddenly drops and that price already appears in the broadcast program, it might be upgraded to a faster disk to account for its importance. With dynamic broadcast disks, it becomes necessary to address the problem of indexing the content so that the client can find out what is available and how long it will have to wait.

The read/write case also introduces a potential problem with respect to data consistency. If values can change at arbitrary points in the broadcast, how can the client guarantee that two values that are read from the broadcast are, in fact, mutually consistent? Traditional database notions of consistency, such as serializable transactions (e.g., as described in [Gray93]) are not likely to be efficiently implementable in the broadcast environment. This opens the question of determining useful models of consistency for broadcast-delivered data and the need for mechanisms to implement them. The Datacycle project addressed this issue by tying the notion of consistency to the period of the broadcast. All data read from the same broadcast period could be assumed to be mutually consistent. Other work in the area, such as that of Barbara and Imielinski [Barb94], has taken the approach of providing currency for individual items, but not providing mutual consistency guarantees across items.

6 Related Work

While no previous work has addressed the integration of multilevel broadcast disks with the related cache management and prefetching techniques addressed here, several projects in mobile databases and other areas have performed related work. Ammar used queueing models and analysis to study related issues in the context of teletex systems [Amma85]. Due to the technology that was assumed in this work the focus was on the properties of broadcast programs in th absence of significant client storage resources. 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 that 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 cellular links as part of our ongoing work.

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. 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.

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 Conclusions

The *Broadcast Disks* project is investigating the use of data broadcast and client storage resources to provide improved performance, scalability, and availability in an increasingly prevalent class of networked applications, namely, those with asymmetric communications capabilities. Wireless networks with mobile clients are one important example of this class of systems. The Broadcast Disks technique exploits communication asymmetry by treating a broadcast stream of data that are repeatedly and cyclicly transmitted as a storage device. The broadcast disk technique has two main components. First, multiple broadcast programs (or "disks") with different latencies are superimposed on a single broadcast channel, in order to provide improved performance for non-uniform data access patterns and increased availability for critical data. Second, the technique integrates the use of client storage resources for caching and prefetching data that is delivered over the broadcast.

In this paper we have presented an overview of the Broadcast Disks technique and have highlighted many of the new research issues that arise when broadcast is used to disseminate data in a distributed information system. We have also described our studies of caching and prefetching policies in a read-only environment. Our current emphasis is on adjusting these strategies to cope with volatile data and changes to the contents and structure of the broadcast. We are also investigating the use of relatively low bandwidth and/or expensive client uplink communication channels for providing feedback to servers and in the construction of hybrid systems that combine traditional request-based data access with data broadcast. We anticipate that data broadcast will become an increasingly important method of delivering data to clients in wired and in mobile wireless environments. Techniques that exploit the power of data broadcasting therefore, have the potential to significantly improve the performance and availability of future distributed computing systems.

Acknowledgements

The authors would like to thank Rafael Alonso for his contributions and ideas during the early phases of this work. This work has benefitted greatly from discussions with Phil Klein, Boaz Patt-Shamir, and M. Ranganathan. Franklin's research was supported in part by NSF grant IRI-9501353 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, an IBM Cooperative Fellowship, and by a gift from Intel Corporation.

References

[Acha95] S. Acharya, R. Alonso, M. Franklin, S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communications Environments", *Proc. ACM SIGMOD Conf.*, San Jose, CA, May, 1995.

[Acha96] S. Acharya, M. Franklin, S. Zdonik, "Prefetching from a Broadcast Disk" *12th International Conference on Data Engineering*, New Orleans, LA, February, 1996, to appear.

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

[Amma85] M. Ammar, J. Wong, "The Design of Teletext Broadcast Cycles", *Perf. Evaluation*, 5 (1985).

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

[Arch86] J. Archibald, J. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model", *ACM TOCS*, 4(4), November, 1986.

[Barb94] D. Barbara, T. Imielinski, "Sleepers and Workaholics: Caching Strategies in Mobile Environments", *Proc. ACM SIGMOD Conf.*, Minneapolis, MN, May, 1994.

[Bowe92] T. Bowen, G. Gopal, G. Herman, T. Hickey, K. Lee, W. Mansfield, J. Raitz, A. Weinrib, "The Datacycle Architecture" *CACM* 35,(12), December, 1992.

[Care91] M. Carey, M. Franklin, M. Livny, E. Shekita, "Data Caching Tradeoffs in Client-Server DBMS Architectures", *Proc. ACM SIGMOD Conf.*, Denver, June, 1991.

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

[Gray87] J. Gray, F. Putzolu, "The Five Minute Rule for Trading Memory for Disc Accesses and the 10 Byte Rule for Trading Memory for CPU Time" *Proc. ACM SIGMOD Conf.*, San Francisco, CA, May, 1987.

[Gray93] J. Gray, A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, San Mateo, CA, 1993.

[Gray94] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, P. Weinberger, "Quickly Generating Billion-Record Synthetic Databases" *Proc. ACM SIGMOD Conf.*, Minneapolis, MN, May, 1994.

[Giff90] D. Gifford, "Polychannel Systems for Mass Digital Communication", Communications of the ACM, 33,(2), February, 1990.

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

[Howa88] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, M. West, "Scale and Performance in a Distributed File System", *ACM TOCS*, 6(1), Feb., 1988.

[Imie94a] T. Imielinski, B. Badrinath, “Mobile Wireless Computing: Challenges in Data Management”, *Communications of the ACM*, Vol. 37, No. 10, October, 1994.

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

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

[Kotz94] D. Kotz, “Disk-directed I/O for MIMD Multiprocessors”, *1st Symposium on OS Design and Implementation*, USENIX, Monterey, CA, November 1994.

[Kist92] J. Kistler, M. Satyanarayanan, “Disconnected Operation in Coda File System”, *ACM TOCS*, 10(1), February, 1992.

[Shek94] S. Shekhar, D. Liu, “Genesis and Advanced Traveler Information Systems (ATIS): Killer Applications for Mobile Computing”, *MOBIDATA Wkshp*, Rutgers Univ., NJ, 1994.

[Wang91] Y. Wang, L. Rowe, “Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture”, *Proc. ACM SIGMOD Conf.*, Denver, June, 1991.

[Wilk90] W. Wilkinson, M. Neimat, “Maintaining Consistency of Client Cached Data”, *Proc. 16th VLDB Conf.*, Brisbane, Australia, August, 1990.

[Zdon94] S. Zdonik, M. Franklin, R. Alonso, S. Acharya, “Are 'Disks in the Air' Just Pie in the Sky?”, *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December, 1994.

Biographies

- *Swarup Acharya* is a Ph.D candidate in the Department of Computer Science at Brown University. His research interests include distributed databases, distributed systems and data management issues in mobile computing. Acharya received his B.Tech(Hons) in Computer Science and Engineering from the Indian Institute of Technology, Kharagpur in 1991 and a Sc.M in Computer Science from Brown University in 1993. He is a member of the ACM and was awarded an IBM Cooperative Fellowship for 1995-96.

- *Michael Franklin* is an Assistant Professor in the Department of Computer Science at the University of Maryland at College Park. His research interests involve the architecture and performance of distributed and parallel database systems. He received the Ph.D. from the University of Wisconsin-Madison in 1993, where he performed research on caching and memory management for client-server database systems and contributed to the design and implementation of the EXODUS and SHORE object storage managers. Previously, he was a member of the Bubba parallel database system project at MCC. He has also been an Invited Professor at INRIA-Rocquencourt. He has served on the program committees of several major database conferences and is an Associate Editor of ACM SIGMOD Record. Dr. Franklin is a 1995 recipient of the National Science Foundation Early Career Development (CAREER) Award.

- *Stanley Zdonik* is a Professor of Computer Science at Brown University, Providence, RI, and has been on the faculty there since 1983. For the past twelve years he has managed the Object-Oriented Database Research Group, which developed the ENCORE and the ObServer object-oriented database prototypes. The group is actively engaged in research in query languages, query processing, and high-performance storage servers for object-oriented databases. Dr. Zdonik has served on the program committees of many major conferences in the areas of databases and object-oriented systems. He has served as an Associate Editor of the ACM Transactions on Office Information Systems and is the founding editor of the ACM SIGPLAN OOPS Messenger. He is currently on the editorial boards of the VLDB Journal, the Journal of Intelligent Information Systems and the Theory and Practise of Object Systems. Dr. Zdonik is co-editor of the books “Readings in Object-Oriented Database Systems” (with D. Maier) and “Implementing Persistent Object Bases” (with A. Dearle and G. Shaw), both published by Morgan-Kaufmann. He was awarded the Office of Naval Research Young Investigator Award in 1986.