To appear in *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994

Are "Disks in the Air" Just Pie in the Sky?

Stanley Zdonik
Dept. of Computer Science
Brown University
Providence, RI 02912

Michael Franklin
Dept. of Computer Science
University of Maryland
College Park, MD 20742

Rafael Alonso
Matsushita Information
Technology Labs.
Princeton, NJ 08540

Swarup Acharya
Dept. of Computer Science
Brown University
Providence, RI 02912

Abstract

Mobile computers and wireless networks are emerging technologies which will soon make ubiquitous computing a reality. In the wireless environment, mobile clients may often be disconnected from stationary server machines or may have only a low-bandwidth channel for sending messages to servers. This environment raises new challenges for the support of database applications for three reasons: 1) the limited storage capacities of mobile machines, 2) the inability to accurately predict the future data needs of many data-intensive applications, and 3) the need to provide clients with new or updated data values in order to ensure consistent data access. One way (perhaps the only way) to address these challenges is to provide the stationary server machines with a relatively high-bandwidth channel over which to broadcast portions of the database. Wireless networks are but one compelling example of the more general class of asymmetric communication environments, *which also includes CATV networks and information distribution services. We have proposed a mechanism called* Broadcast Disks *to provide database access in asymmetric communication environments. By appropriate organization of the broadcast program, an arbitrarily fine-grained memory hierarchy can be created. Such a hierarchy raises fundamental new issues for pre-fetching and client cache management. In this short paper we present a brief overview of asymmetric environments, our novel approach to organizing broadcast disks and some initial simulation results.*

1 Introduction

Many new networked applications have the property that the communication channel between the client and the server is *asymmetric*. By this, we mean that the available bandwidth from the server to the client is significantly greater than the available bandwidth from the client to the server. Some examples of this situation include:

- wireless mobile clients that are disconnected from their server,
- a cable television hierarchy (or, a video on demand service) with high downstream bandwidth and very limited upstream capacity,
- a very large number of clients doing similar tasks in which each can be allocated only a small amount of client-server bandwidth (e.g., stock market trading)
- an application with extremely volatile information (e.g., air traffic control in which old data can be dangerous),

We are currently investigating a new technique that we call *broadcast disks*(BD)[AAF94]to provide clients with data in a way that reduces the need for relatively expensive client to server communication. The basic idea of a broadcast disk is that a stationary server continually broadcasts data which is of interest to a set of possibly mobile clients. This data is broadcast repeatedly, thus making it appear to be rotating; the broadcast can be thought of as a spinning disk with the clients' receiver as the analog to the disk head. In effect, this broadcast data is an intermediate level in the memory hierarchy between the client and server; when the client does not have a desired item locally (i.e., in its memory or disk), it obtains the page from the BD (if possible), thereby avoiding the need to contact the server.

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 ([SKM93], [ImB94]). 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, it is important to note that such disconnection is simply an extreme case of asymmetric communication. Similar issues can arise in the other applications listed above — even in 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)[ShL94]. 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 is around 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 to transfer data — individual clients query the server or the server broadcasts data, the latter is inherently better suited to the ATIS environment. A system based on querying suffers from scalability and problems of disconnection. Besides, given the public nature of the information and typically very similar queries, broadcasting is the ideal choice. If the broadcast bandwidth is severely limited, only the dynamic information would be sent. The static information could be supplied by the driver on CD-ROM.

With current mobile hardware, listening to the broadcast requires CPU cycles as well as battery power. This may not be a problem for all mobile applications. For example, in ATIS a car battery is more than adequate. For those applications where battery power must be conserved, we expect that direct hardware support will become available once the analogy of repetitive broadcast to a disk becomes more widely accepted. We envision a device that acts much like a disk controller that can run independently of the processor and that is reasonably power efficient. This *broadcast disk controller* will monitor the broadcast for interesting events. This should ease the problem of power consumption and *tuning* time[IVB94b].

1.2 Broadcast Disks

Data broadcast has been investigated by other researchers in both wired [BGH92], and wireless([IVB94b], [IVB94a]) environment. Our work differs however, in that we have generalized the notion of "broadcast disks" to include an arbitrary number of "disks" of varying sizes and speeds, that can be superimposed on the broadcast medium. This approach allows the broadcast program to be tailored to improve performance for non-uniform access patterns — popular or important data items can be broadcast more frequently than other items.

The multiple-disk approach to data broadcast poses many challenging database 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 order in which data is broadcast is crucial for 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 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 Broadcast Program Structure

2.1 Characteristics of Broadcast Disks

The simplest structure for organizing the data broadcast would be a single, huge disk (i.e., a flat broadcast) containing all of the data to be broadcast (this is the approach of earlier work in the area). In this scenario, all items are broadcast with equal frequency, regardless of their importance to the client community. We advocate an alternative broadcast organization; data is placed on multiple disks of varying

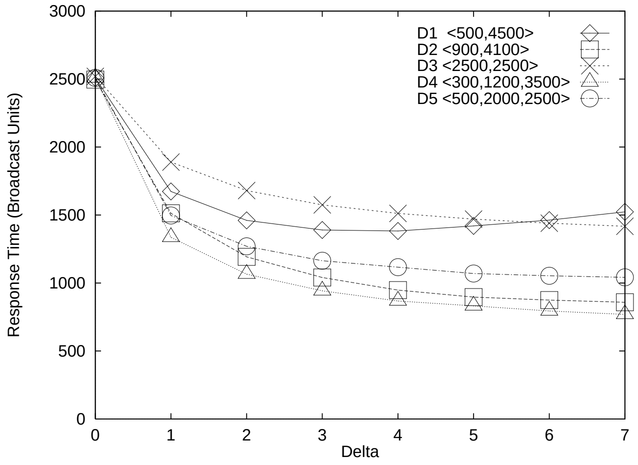

Figure 1: **Response Time, Cache Size = 1**

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" [GrP87].

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* object, the expected delay in accessing the page 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.

Assuming that there exists only a single broadcast channel, a multi-disk organization would require some form of interleaving between disks with different broadcast frequencies. We have developed a technique to do this multiplexing in a manner that produces broadcast programs with several desirable properties. First, repeated broadcasts of the same item are evenly spaced, thereby minimizing the degradation in expected access time caused by variance in the interarrival rate of copies of an item. Second, the broadcast is periodic, which provides a natural boundary for consistency guarantees in the presence of updates, as well as for re-organizing the program. Finally, the technique is flexible, allowing the creation of any number of disks whose relative speeds can vary in any proportion. The details of this technique are beyond the scope of this short paper, but can be found in [AAF94].

Several other issues must be addressed in the design of the broadcast disk approach. Because unlike traditional client-server systems, data is disseminated rather than requested, data items in the broadcast must be self-identifying so that clients can determine which items are arriving on the broadcast. Furthermore, if the content and/or structure of the broadcast is allowed to change dynamically, then some form of indexing will be needed to enable clients to adjust their caching and prefetching policies to support correctness and efficient access (Indexing policies for flat broadcast programs have been investigated in [IVB94b] and [IVB94a]). We briefly discuss the dynamic aspects of broadcast organization in Section 5.

2.2 Example Performance Results

We have performed a detailed simulation study of the properties multi-layered disks. Descriptions of these results for different client and server configurations are beyond the scope of this paper and are available in [AAF94]. Here, we present a single graph from that study in order to give a flavor of the tradeoffs involved in a broadcast environment. Due to space constraints, we provide only limited information about the simulation model. Figure 1 shows the performance of a disk-less client with a cache size of one (i.e., the client retains only the currently accessed page) for various broadcast disk configurations.

The y-axis represents the client response time and the x-axis represents Δ(Delta), a metric used to measure the relative spin speeds of the various disks. Let there be N disks numbered 1(fastest) to N(slowest). For any given Δ, the ratio of the spins of the i^{th} disk to the slowest disk N is given by $(N-i)\Delta+1$. When $\Delta = 0$, the broadcast is flat; all disks spin at the same speed. The graph represents a broadcast database size of 5000 pages. The pages of interest to the client, the client *access range*, are the first 1000 pages, accessed following a Zipf distribution.

Figure 1 shows how a client which does no caching performs for various two and three disk configurations. The legend shows the configuration of various broadcasts– the numbers following the mnemonic represents the size of each disk, in decreasing order of spin. As would be expected, for a flat disk($\Delta = 0$), the response time is 2500 pages — half the database size. As Δ increases, all the disk configurations provide better performance than a flat disk eventually flattening out around a Δ of 3 or 4. We explain the

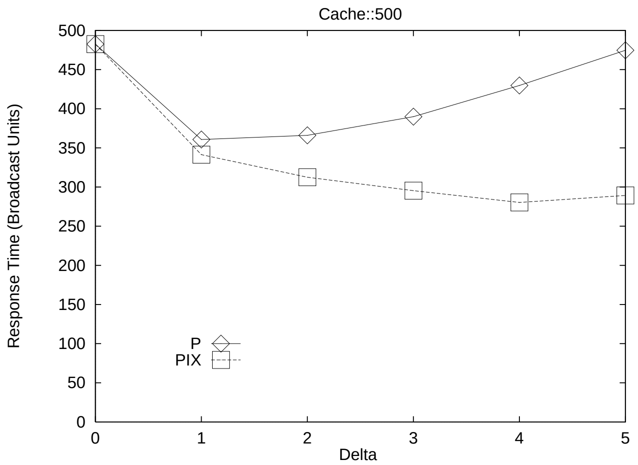

Figure 2: $\mathcal{P}$ **vs.** $\mathcal{PIX}$ **with varying** Δ

intuition behind the improved performance using the first disk configuration, D1. In D1, 500 pages (which are also the client's hottest) fit on the fastest disk and the rest of the access range is on the second slower disk. As Δ increases, the hot pages are brought closer to the client. However, beyond a point ($\Delta > 4$), the penalty of accessing pages on the slower disk (which get pushed farther and farther away), causes degradation of performance. This points to an inherent tradeoff of broadcast disks — pages can be brought closer to the client by spinning them faster but it is at the expense of delaying access to other pages; the key is to exploit this tradeoff. The remaining disk configurations represent a pair of two and three-level disks. As can be seen, adding an extra level can improve performance by allowing the broadcast to be better matched to the client access needs, however, the addition of another level does not necessarily imply better performance.

3 Client Cache Management

A new set of issues arise once client caching is taken into consideration. Caching in a typical wired environment is a means to increase performance by keeping objects of interest in the fastest accessible memory (client cache). While increasing performance is also the goal of caching in the broadcast environment, there is a key difference. Consider a multi-layered disk with some objects appearing in the broadcast more often than the others. Because the broadcast structure is determined by taking into account the access needs of all of the clients that may listen to the broadcast, it is very unlikely that the chosen broadcast program matches any one client's access needs exactly. For example, an item that is *hot* to a client may be placed on a slow spinning disk because it may not be important for most other clients. Similarly, clients may find some of their *coldest* items in the faster disks if those objects are *hot* to other clients.

If a client is not able to cache data that it obtains from the broadcast, then it has no alternative but to obtain its needed items from the broadcast as they go by. Such a client will be adversely affected by the mismatch between its access needs and the server broadcast program. However, if the client has a cache, it is no longer at the mercy of the server's broadcast structure. By choosing to selectively cache items (and thus not needing to go to the BD), it effectively is able to modulate its access to the broadcast in order to more closely match the access pattern for which the server has generated its broadcast. Thus, the role of the client cache in a broadcast environment, is to diminish the client-server differential. To this end, a client may in fact, purge some objects from the cache even though they may be *hot* and hold onto objects which may not be greatly in demand. This is a fundamental difference compared to traditional client-server systems.

The client cache manager has to selectively cache items from the broadcast so as to minimize the mismatch between its access pattern and server's broadcast program. In particular, the client must use its cache to minimize the delay caused by waiting for pages that are broadcast on the slower disks. Thus, the client cache replacement policy must be based not only on the expected probability of access for each item, but also on the cost of fetching the items back once they are replaced. An efficient heuristic which reflects the cost of replacement of an item is the ratio of the probability of access(P) and its broadcast frequency(X). This ratio (P/X) is called the *pix* value (P Inverse X). The $\mathcal{PIX}$ replacement algorithm ejects the item in the cache which has the lowest *pix* value. To see the potential performance advantages of cost-based caching over caching techniques which use probability alone to replace pages, consider Figure 2. It shows the performance of $\mathcal{PIX}$ and $\mathcal{P}$, a cache replacement policy which evicts the cache-resident page with the lowest probability of access. The graph shown is for a cache size of 500 pages in a situation in which there is some difference between the broadcast program and the client access pattern. As Δ increases, response time for $\mathcal{P}$ begins to increase quickly whereas $\mathcal{PIX}$ falls to about half the value of that of the flat disk (at Δ=4) before rising again. This is because unlike $\mathcal{P}$, $\mathcal{PIX}$ is sensitive to Δ which reflects the frequency of broadcast.

The insight gained from the investigation of $\mathcal{PIX}$ has led to the development of a more practical (i.e., implementable) algorithm based on LRU. This new algorithm, called $\mathcal{LIX}$ uses a separate LRU chain for each disk of the multi-disk broadcast. $\mathcal{LIX}$ computes a cost value similar to $\mathcal{PIX}$ but only for least recently used item of each chain, and chooses the one with the lowest value as the replacement victim. The LRU chains are dynamically sized — an item that is read into the cache is placed on the LRU chain for its disk, regardless of the chain from which the replacement victim that made room for it was chosen. A detailed analysis and description of these algorithms is presented in [AAF94].

Finally, it should be noted that the introduction of client caching also impacts the generation of broadcast programs at the server. The server must adapt its broadcast to account for the fact that a client caches a portion of its access range. Consequently, the server's strategy is to broadcast these cached pages less often and thus, allocate lower bandwidth for these items which are accessed from the cache and not from the broadcast.

4 Prefetching

In the previous sections we described a scheme in which clients obtain data from the broadcast only as the result of a cache fault for an application running at that client. An alternative client-side approach is to bring data from the broadcast into the client cache in advance of any requests. This opportunistic method of managing the client cache is referred to as *prefetching*. The decision about which pages to prefetch is made at the client as pages pass by on the broadcast. The idea is simply to replace currently cached pages with broadcast pages that are more valuable. In this way, the client cache will automatically increase in temperature with no intervention from the application.

The work described in the remainder of this section is preliminary results of a study that we are conducting into the use of our style of broadcast disk to support prefetching. The discussion is meant to be suggestive and is not conclusive.

4.1 Latency-Based Prefetch

As in the $\mathcal{PIX}$ cache replacement policy described earlier, we need a metric that reflects the relative value of a page. One such promising metric is the product of the probability of access p of a page P and the amount of time t until P appears in the broadcast again. Our idealized algorithm prefetches a page P if its pt value is greater than the pt value of some page in the cache. The cached page with the lowest pt value is evicted. We call this the $\mathcal{PT}$ prefetch algorithm.

$\mathcal{PT}$ is an ideal policy in that, as it stands, it is not practical to implement. First, it needs knowledge of the probability of access for both cached and non-cached pages. Moreover, it is enormously expensive - it requires an O(Cache Size) computation for every page in the broadcast. We show it as a proof of concept. We are presently designing a practical approximation to $\mathcal{PT}$ much as we did for $\mathcal{PIX}$ and $\mathcal{LIX}$.

To get a sense for the utility of prefetching, we compared $\mathcal{PT}$ to $\mathcal{PIX}$ which is also an ideal policy. Consider Figure 3. It shows the client response times for the two algorithms with varying Δ for the two disk broadcast program D1 from Figure 1. To make simulations feasible, we cut down the space by a factor of 5 from the space in Figure 1. Consequently, the broadcast database size is 1000 pages and the client access range is 200 pages. The graph represents a client with a cache size of 100. As the graph shows, $\mathcal{PT}$ cuts the response time by about a fifth compared to $\mathcal{PIX}$ justifying our intuition.

$\mathcal{PIX}$ could be used as the basis for prefetch as well. Pages would be prefetched when their *pix* value exceeds the *pix* value of a page that is in the cache. When this method was tried, it performed exactly the same as the non-prefetch version of $\mathcal{PIX}$. Since $\mathcal{PIX}$ is based purely on static values, it will tend to load the cache up with the pages with the highest *pix* values. When a new page is accessed, the page with the lowest *pix* value will be ejected, only to come back quickly from the prefetch. In other words, the cache will contain the same pages in both cases. In order to do better, the prefetch scheme must take advantage of some dynamic access of the broadcast, for example, the current point of broadcast.

Ignoring access probabilities, $\mathcal{PT}$ replaces pages which will appear again soon in favor of pages which are far away in time. $\mathcal{PIX}$ is based purely on frequency and will not distinguish the case in which the page of interest just passed by from the case in which the same page is coming right up. Thus, $\mathcal{PT}$ is more accurate than $\mathcal{PIX}$.

We have observed in other experiments that $\mathcal{PT}$ performs best when the hot pages are not clustered together in the broadcast. In a clustered configuration, these pages will all have approximately the same pt values at all times making them all equally likely to persist in the cache. In fact, since they are hot they will certainly persist until just before they arrive again in the broadcast, at which time, they might be ejected for a short time, only to be reintroduced when they arrive. Thus, as in $\mathcal{PIX}$, the hottest pages will spend most of their time in the cache independently of the current position of the broadcast cycle. As the hottest pages are distributed more evenly on the broadcast disks, the benefit of $\mathcal{PT}$ begin to show.

The simulations are performed as follows: we start with all the pages of interest clustered together. Then using a de-clustering technique, whose details we do not go into in this paper, we randomize the placement of a portion of the client access range. The graph of Figure 3 represents the case where 70% of the pages have been randomly placed. Note that in real life given the differing access patterns of various clients, the pages of interest for a particular client would very likely be non-contiguous in the broadcast.

The scheme described in the next section also shows the benefit derived from a uniform distribution of

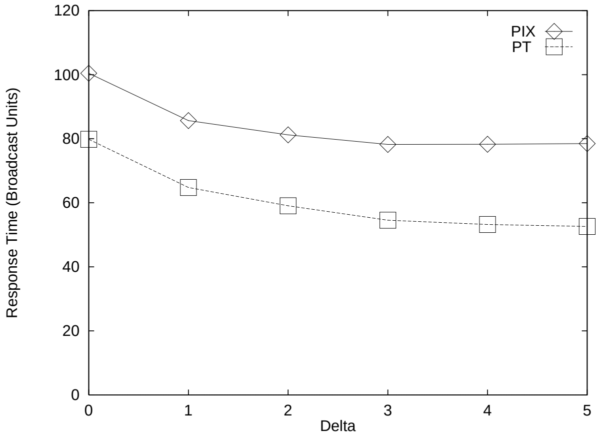

Figure 3: **Comparison of $\mathcal{PT}$ and $\mathcal{PIX}$ for a two-disk broadcast**

pages in the broadcast coupled with prefetching. The benefit comes from using prefetch to effectively increase the size of the cache by decreasing the average latency for non-cached items.

4.2 Tag-Team Caching

In the previous section, prefetching is done by dynamically computing a goodness measure. In contrast, we are also investigating a scheme in which a pattern of probability distributions on the broadcast disk is exploited to improve the expected performance. We call this technique *tag-team caching*.

This approach to prefetching is best explained by a simple example. Assume that we have two pages x and y that each have a probability of access of 0.5 and that the broadcast is a flat disk. Further, let us position x and y be 180 degrees apart on the broadcast disk (exactly opposite each other). Tag-team caching would read x into the cache until y came by at which point y would replace x in the cache. When x came by again, it would replace y in the cache, and so on. We will compare this strategy to that of simply reading x into the cache and accessing y from the broadcast. We will call the later case *simple caching*.

For the simple caching case, the cost of accessing x is zero, while the cost of accessing y is half the rotational period. Since both of these events have a probability of 0.5, the total expected cost of accessing x or y is one quarter of a revolution.

For the tag-team caching case, the cost of accessing x when x is in the cache is zero and the cost of accessing x when x is not in the cache is one quarter of the period. Since x is in the cache half of the time, the average cost of accessing x is one eighth of a period. The expected cost of accessing x or y is the probability of accessing x times the average cost of accessing x plus the probability of accessing y times the average cost of accessing y This is one eighth of the period as opposed to one quarter of a period for the simple caching case.

This result can be generalized to multiple pages with different sized caches. We are currently studying this case in detail and intend to compare this more static style of prefetching with results of a dynamic scheme like $\mathcal{PT}$.

5 The Read/Write Case

An important extension of our existing research is to address the case in which the broadcast can change. The ability to do dynamic broadcasts opens up a large numbers of applications like ATIS, stock market monitors, etc. which deal with inherently volatile data. There are at least three different ways by which a broadcast can change. One, the data value of objects being broadcast changes. Two, the layout of the broadcast is altered wherein pages are shuffled between different disks. Finally, the content of the broadcast disk itself may change bringing in new pages and throwing out existing pages.

A value in the broadcast will change when some connected client commits an update. Alternatively, the broadcast application may itself be generating 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[BaI94].

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 BD may become interesting when it's price changes. Therefore, at that point, it should be moved onto the BD, and perhaps displacing something else. Alternatively, if the stock price suddenly drops and is already being broadcast, 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 this 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 BD are, in fact, self-consistent?

Our current scheme assumes that it is easy to know the frequency of broadcast of a page because once the client listens to a complete cycle, the observed frequencies will not change. For the read/write case, this will no0 longer be true. The server will have to broadcast some form of summary information (e.g., an index) along with the data.

We also assume that if a page is in the client cache, then it is up to date. Once volatile data is broadcast, we have to address the issue of cache invalidation or update propagation in a broadcast setting([BaI94], [WiN90], [CFL91], [WaR91]).

6 Related Work

Several projects in mobile databases and other areas have performed work that is related to our research on broadcast disks. The Datacycle project at Bellcore ([HGL87], [BGH92]) investigated using a very high bandwidth optical network as a repetitive broadcast medium for performing associative searches and continuous queries on the broadcast data. Datacycle broadcast data using a flat disk approach and so did not address the multi-level disk issues or the caching or prefetch issues.

More recently, the mobile computing group at Rutgers has investigated techniques for indexing broadcast data ([IVB94b], [IVB94a]). 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 [IVB94a] 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. Very recently, scheduling for satellite-based broadcast-based file systems has been proposed in [Chi94].

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, tertiary mass storage [AnH93], magnetic disks [AkS92] and even in client-server storage systems [FCL92] have used cost-based replacement in some form or other.

7 Conclusions

Many new applications have the property that the communication channel between the client and the server is *asymmetrical*. One extreme example of this is that of disconnection of a mobile host in a wireless network where there is only a one-way communication channel from the server to the mobile host. This paper highlights a technique which we call *Broadcast Disks* which tries to alleviate the problem of very low bandwidth (and/or expensive) uplink costs by broadcasting data to clients. Broadcast disks introduce many new research problems and we highlight some of the issues involved. The paper also gives some initial simulation results which show the efficacy of our approach to the design of broadcast disks.

There is clearly an issue of bandwidth availability in a mobile wireless environment. The lower the bandwidth, the higher the latency for a given amount of data. For modest sized data requirements, as in ATIS applications, the bandwidth may be sufficient. Also, for disconnected applications, one might be willing to pay the price of high latency since the alternative is no access at all.

We believe that judicious application of this technology coupled with a careful understanding of the design space will make *disks in the air* more than just *pie in the sky*.

Acknowledgements

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

[AAF94] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments," Dept. of Comp. Science, Brown University, CS-94-43, (also, Univ. of Maryland, Dept. of Comp. Science, CS-TR-3369), October 1994.

[AkS92] S. Akyurek and K. Salem, "Placing Replicated Data to Reduce Seek Delays," *Proc. USENIX File System Conference* (May 1992).

[AnH93] C. Antonelli and P. Honeyman, "Integrating Mass Storage and File Systems," *Proc. 12th IEEE Symposium on Mass Storage Systems* (1993).

[BaI94] Daniel Barbara and T. Imielinski, "Sleepers and Workaholics: Caching Strategies in Mobile Environments," *ACM SIGMOD*, Minneapolis, Minnesota (1994).

[BGH92] T. G. Bowen, G. Gopal, G. Herman, T. Hickey, K. Lee, W. Mansfield, J. Raitz, and A. Weinrib, "The Datacycle Architecture," *CACM* 35(12) (December 1992).

[CFL91] M. Carey, M. Franklin, M. Livny, and E. Shekita, "Data Caching Tradeoffs in Client-Server DBMS Architectures," *ACM SIGMOD*, Denver, CO (June, 1991).

[Chi94] T. Chiueh, "Scheduling for Broadcast-based File Systems," *Proc. of the Mobidata Workshop*, Rutgers Univ., New Brunswick, NJ (Nov 1994).

[FCL92] M. Franklin, M. Carey, and M. Livny, "Global Memory Management in Client-Server DBMS Architectures," *Proc. 18th VLDB*, Vancouver, Canada (August, 1992).

[GrP87] J. Gray and F. Putzulo, "The Five Minute Rule for Trading Memory for Disc Accesses and the 10 Byte Rule for Trading Memory for CPU Time," *ACM SIGMOD*, San Francisco, CA (1987).

[HGL87] G. Herman, G. Gopal, K. Lee, and A. Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems," *ACM SIGMOD*, San Francisco, CA (May 1987).

[ImB94] T. Imielinski and B. R. Badrinath, "Mobile Wireless Challenges in Data Management," *Comm. of the ACM* 37(10), (October 1994).

[IVB94a] T. Imielinski, S. Viswanathan, and B. Badrinath, "Power Efficient Filtering of Data on Air," *Proc. EDBT Conf.*, Cambridge, U.K. (March 1994).

[IVB94b] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Energy Efficient Indexing on Air," *ACM SIGMOD* (1994).

[SKM93] M. Satyanarayanan, J. J. Kistler, L. B. Mummert, M. R. Ebling, P. Kumar, and Q. Lu, "Experience with Disconnected Operation in a Mobile Computing Environment," *USENIX Symp. on Mobile and Location Ind. Computing* (Aug 1993).

[ShL94] S. Shekhar and D. Liu, "Genesis and Advanced traveler Information Systems(ATIS): Killer Applications for Mobile Computing?," *Proc. of the Mobidata Workshop*, Rutgers Univ., New Brunswick, NJ (Nov 1994).

[WaR91] Y. Wang and L. Rowe, "Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture," *ACM SIGMOD*, Denver, CO (June, 1991).

[WiN90] W. Wilkinson and M. Neimat, "Maintaining Consistency of Client Cached Data," *VLDB*, Brisbane, Australia (August, 1990).