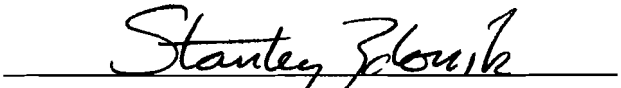
Client-Server Performance Evaluation in Pushed-based Systems

Jie Cui

Department of Computer Science
Brown University

Submitted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Computer Science at Brown University

November, 1998



Professor Stanley Zdonik
Advisor

Abstract

As the asymmetric communication environments (wireless, satellite, ...) emerges in recently years, there are increasing interests in studying the push-based client server models, in contrast to the traditional emphasis on pull-based systems. In a few previous papers, Professor Zdonik and his collaborators: Acharya and Franklin et al.[1, 2, 4], have proposed a novel push-based system, which they called "broadcast disks", and they have done software simulations to validate the performance of the proposed broadcast methods as well as to studying the performance of various cost based caching management policies.

In this master's thesis, our purpose is mainly to run the experiments on a prototype system that consists of four PCs with Pentium processors. The experimental results faithfully confirmed the conclusions drawn by simulation.

This thesis has two main contributions:

1. Derive the mathematics for designing the optimal broadcast page frequency, by solving a constraint optimization problem. We found that the optimal frequency of broadcast should be proportional to the square root of the access frequency averaged over all clients.

2. Programming and experiments on the prototype system. The experiment largely confirm the advantages of the push-based system in the asymmetric communication environments.

Acknowledgments

I'd like to thank for professor Stanley Zdonik for encouraging me to participate this research project, and for technical advice and his patience. I am also grateful to Rahul Bose and Swarup Acharya for their discussion, advice, and assistance throughout the experiments.

Through the work of this thesis, I have learnt the broadcast disks system, and did experiments on a prototype system. The research is of great values to my professional career. During the work, I also learn to formulate a problem using mathematical tool, and learn to compare various parameters and management policies carefully in order to draw firm conclusions.

The research is supported by an ONR grant No. N00014-91-J-4085 and an Intel gift to Professor Zdonik.

1 Introduction

The past decade has witnessed an explosion in data communication through the growth in the Internet, cable networks, mobile computers and satellite communication. The advances in data communication together with the ever-growing computing power pose interesting challenges in evaluating traditional assumptions in designing large scale distributed systems. In a series of papers[1, 2, 3, 4], Acharya, Franklin and Zdonik have identified a set of emerging applications domains, which they called the *Asymmetric Communication Environments.* In an asymmetric communication environment, the downstream communication capacity (from the servers to the clients) is much greater than the upstream communication capacity (from the clients to the servers). This asymmetry can often be caused by 1). bandwidth asymmetry in the physical network, an extreme example is the uni-directional communication from the server on the satellite to to mobile clients on the ground, or 2). by the unbalanced data volume up/down the channels. Examples of the asymmetric communication environment include: 1). wireless networks where the servers are stationary and the clients are mobile, 2). Information dispersal systems, such as stocks, traffic, weather, batter fields and so on.

Furthermore, Acharya, Franklin and Zdonik have proposed a novel dissemination based approach for data distribution in the context of client-server systems, and they also proposed a new architecture called *broadcast disks* to address the data distribution issues in the asymmetric communication environment. In contrast to the traditional *pull-based* methods, the broadcast disks are *push-based* architecture, where a server repeatedly broadcast data through a communication channels, and the clients retrieve pages as they appear in the channels. The architecture of the broadcast disks is illustrated in figure 1.

As shown in figure 1 [1], the design of the broadcast disks consists of two parts:

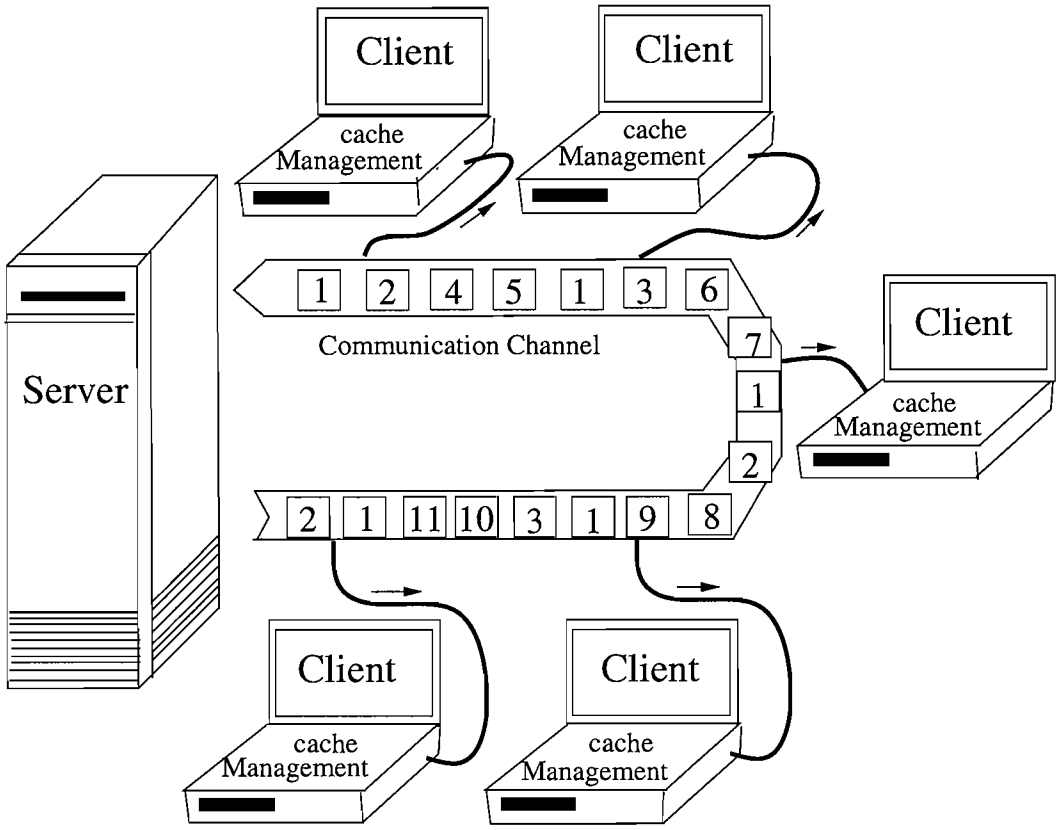


Figure 1 The architecture of the multiple broadcast disks– a push-based client-server system.

1). design of broadcast disks on the server side, and 2). cache management on the client side.

1. **Broadcast disks arrangement.** One simplest way of designing the broadcast program is the so called *flat disk*[1]. In this method, a single disk is adopted in the server, and all data pages are broadcast periodically at the same frequency. Thus if a client is missing a page, it has to wait half of the period on average to catch it from the broadcast channel. To improve the response time, a good broadcast system must take into account the access frequencies of the pages in the database, so that the "popular" pages are broadcast more frequently. One underlying hypothesis is that the server knows the access pattern in advance, and this pattern is stable over a certain period of time. This assumption is valid when there is a large population of clients, where the statistics fluctuation is small according to some large deviation theory in statistics, for example, by law of large numbers.

In section (2.1), we provide the mathematical analysis for designing the optimal broadcast frequency for each page, given the client access patterns. The conclusions are the followings.

1. Each page should be broadcast periodically with zero variance in the inter-arrival time.

2. The broadcast frequency of each page should be proportional to the *square root* of the access frequency averaged over the client population.

The broadcast disks system proposed by Acharya, Franklin and Zdonik[1] addressed these issues. They first discretized the broadcast spectrum into a finite number of frequency buckets. All pages in the same bucket are broadcast in the same speed. Intuitively, pages in the same frequency bucket are loaded in a disk, which spins at a certain frequency. As the total bandwidth is fixed, the pages in different disks inter-weaves carefully into the channel.

However, the broadcast frequency of each page was set to be equivalent to the demanding frequency, which is suboptimal according to our analysis in section (2.1). Nevertheless, the multi-disks system demonstrate better performance than the flat disk.

2. **Client cache management**. Caching is a simple technique inherited from the hierarchic memory management in computer architecture. In a client-server DBMS system, the caching techniques is adopted in the client side to store the most frequently accessed pages in a cache in order to improve the response time in a given application. In a broadcast client-server system, the major purpose of the caching management is to overcome the deviation between 1). the page broadcasting frequency adopted by the server, and 2). the page access frequency of an individual client.

As the client cache is much smaller than the size of the database in the server side, the major issue of research in the literature is the *page replacement policy.* In the new asymmetric communication environment, three kinds of page replacement policy are considered in [5].

1. Standard passive cache management policy, such as First In First Out (FIFO), Least Recently Used (LRU), and Least Frequently Used (LFU). In particular, LRU is chosen for comparison for ease of implementation.

2. Cost based caching. Policy that accounts for the push-based communication. *PIX, LIX* and so on. In the broadcast disk systems, the cache policy should not only consider the access frequency of the client, but also take the broadcast frequency into account. Thus a few novel page replacement schemes have been proposed by Acharya etc[1]. Figure (2) displays the preference of pages for cache memory in a broadcast systems.

3. Prefetching. Prefetching is a technique that the client requests a page ahead

of time. To do so, a client must sample every page in the channel and compute whether the page is prefetch-worthy. If yes, the new page will replace the least important page in cache. Thus the computation must be efficient in order to keep up with the broadcast speed. It can potentially improve the performance in two ways: 1). increasing the hit rate, 2). reducing the latency in a miss.

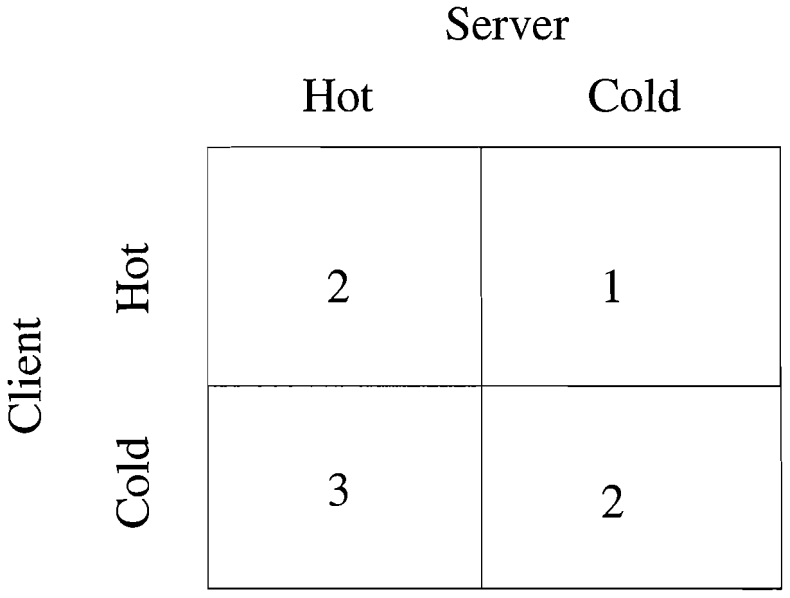


Figure 2 The priorities of pages for cache memory in a broadcast system. The top priority is assigned to pages that are cold in the server/channel and hot in the client access frequency. The figure is modified from [5]

The tasks of this master thesis is to evaluate the performance of the broadcast disks in a prototype system. The results of our experiments are plot in section (4). These results faithfully confirmed the major conclusions drawn in previous paper by Acharya, Franklin and Zdonik[1, 2, 3, 4]. Thus they validated that the simulated environment can characterize the properties of push-based systems.

The thesis is organized as follows. We first discuss the mathematics in designing the broadcast programs in section (2.1). Then we briefly study the broadcast model for the server and the broadcast program in section (2.2), and the client models

and cache management policy in section (2.3). Section (3) describes the prototype system for experiments. Section (4) reports the experiment results. We conclude the thesis by a short discussion in section (5).

2 The Broadcast Disks: a Pushed-based Data Communication System

We discuss the issues in structuring the broadcast disks in this section. We start with a mathematics study of the broadcast frequency selection, which is followed by discussion on the design of server and clients.

2.1 The mathematics in selecting broadcast frequency

In this subsection, we derive two mathematical theorems for designing the optimal broadcast frequency in order to minimize the total response time at the clients.

Fixing the bandwidth of the communication channel and the size of the database, the task of designing the broadcast disks system can be posed as an optimization problem, which allocates the bandwidth to the data pages in order to achieve optimal response time.

To fix notation, let M be the number of pages in the database, and K the total number of clients in the broadcast system. Let F_{ji} be the frequency at which the j-th client requests the i-th page, and f_i be the broadcast frequency of page i at the server.

When the broadcast system reaches its steady state, at each client the average waiting time for page i is simply $\frac{1}{2} \cdot \frac{1}{f_i}$. Thus the total average waiting time for all client is

$$\frac{1}{K} \sum_{i=1}^{M} \sum_{j=1}^{K} \frac{F_{ji}}{2f_i} = \sum_{i=1}^{M} \frac{F_i}{f_i}$$

where $F_i = \frac{1}{K}\sum_{j=1}^{K}\frac{F_{ji}}{2}$ is the access frequency of page -i -averaged across the client population.

Furthermore, as the bandwidth is fixed, we have

$$f_1 + f_2 + \cdots + f_M = B$$

with B being the total bandwidth of the channel. Therefore we pose the broadcast design as the following optimization problem:

$$(f_1, f_2, ..., f_M)^* = \arg\min \sum_{i=1}^{M} \frac{F_i}{f_i}$$

$$\text{Subject to constraint} \quad f_1 + f_2 + \cdots + f_M = B$$

Solving the above constrained optimization problem by Lagrange multipliers, we need to minimize

$$\sum_{i=1}^{M} \frac{F_i}{f_i} + \lambda(\sum_{i=1}^{M} f_i - B)$$

Taking derivative over f_i and setting it to zero, we obtain

$$-\frac{F_i}{f_i^2} + \lambda = 0$$

$$f_i = \sqrt{\frac{F_i}{\lambda}}$$

Plugging the above f_i to the constraint equation to solve for λ–the Lagrange multiplier, we have

$$f_i = \frac{B\sqrt{F_i}}{\sum_{i=1}^{M}\sqrt{F_i}} \tag{1}$$

Equation (1) tells us that

Theorem 2.1 *The broadcast frequency for each page should be proportional to the square root of the average access frequency.*

In a more general case, one may assign different priority to different clients in calculating F_i, so that the access patterns of high priority client have bigger weights in the optimization criterion.

The above conclusion is derived based on the assumption that each page is broadcast periodically, i.e. no variance in the inter-arrival time. To release this constraint, one may allow the variation in inter-arrival period.

For example, page i is broadcast at interval $\Delta_{it} = \frac{1}{f_{it}}$ at time t, and one constraint is that

$$\frac{1}{L}\sum_{k=1}^{T} f_{it} = f_i,$$

i.e. each page has a fixed broadcast frequency, and T is the time length that the system runs.

Thus the generalized problem is to compute the optimal broadcast frequency $f_{it},\ i = 1, 2, ..., M, t = 1, 2...., T$ to minimize the response time:

$$\{f_{it},\ i = 1, 2, ..., M, t = 1, 2...., T\}^* = \arg\min \frac{1}{T}\sum_{i=1}^{M}\sum_{t=1}^{T}\frac{F_i}{f_{it}}.$$

It is trivial to prove that

$$f_{i1} = f_{i2} = \cdots = f_{iT}$$

that is

Theorem 2.2 *The variance of the broadcast frequency for each page should be zero.*

In a more general setting, we should take into account the cache management policy in the optimization problem, and it seems possible to analyze the performance explicitly using theories in discrete event stochastic systems. However, this study is beyond the scope of this master thesis.

2.2 *Server: A Simplified Model for the Broadcasting Environment*

In real push-based communication systems, the number of clients can be very large, however, it is impractical in a laboratory to test a broadcast systems with thousands of computers running altogether. Indeed, there seems no need for such brute-force hardware simulation. Instead, the performance of the client population can be assessed through parameters in designing the server and clients. The parameters chosen in the simulator are

1. A client in the testing system only access to a subset of the pages in the database. This reflects the fact that the server broadcast for many clients.

2. Adding noise perturbations to the broadcast frequency, and thus create deviations between the broadcast frequency and the demanding frequency. This reflects the fact that the server can only match the demand of a client partially.

In summary, the server is modeled by the following 6 parameters.

1. $ServerDBSize$ – This is the total number of distinct pages in the database to broadcast.

2. $NumDisks = N$ – the is the number of disks in the server, and it reflects the precision of the server frequency in matching the client access frequency. The more disks we have, the more freedom we have in matching the access patterns. However, after a certain limit the improvement will become less and less as N grows.

3. $DiskSize_i$, $i = 1, 2..., N$ – these are the page numbers for each disk.

4. Δ – the broadcast frequencies of the disks are assumed to be a linear model with Δ being the slope. As shown in figure (3)

5. *Noise* – this controls the degree of disagreement between the needs of the client and the server. When $Noise = 0$, the broadcast frequency is proportional to the access frequency of a client. Large *Noise* reflects the reality of mismatch between server and clients. For example, the access frequency of a client may change dynamically, and there are many clients in a system.

6. $Offset$ – This takes into account the effects of cache usage in the client side. If the client always keeps the hottest pages in its cache, then the server will move these pages to the slowest disk in order to save slots for other pages. The size of the offset is in general in the order of the cache size. It moves the $Offset$ number of pages from the fastest disk to the slowest disk.

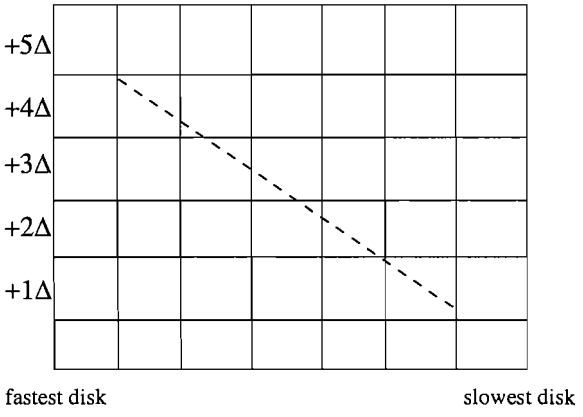


Figure 3 The broadcast frequency is modeled as a linear equation as shown by the dashed-line, the speed of the broadcast disk increases Δ each time from the slowest disk to the fastest disk.

In summary, the broadcast pages are generated as follows. First sort the pages sequentially from hottest to coldest. Second, divide the queue into $NumDisks$ disks. Third, shift the pages by $Offset$ so that the hottest $Offset$ pages is moved to the end of the queue. Fourth, for each page in the queue, a coin weighted by *Noise* is tossed. If based on the coin toss, this page i should swap. Then a disk $d \in \{1, ..., N\}$ is chosen at uniform distribution, page i is then switched with a

random page at disk d. Intuitively, this noise perturbation has larger impacts to the smaller disks, usually the faster disks. As each disk has a equal chance to be chosen, and the pages in the smaller disks are more likely to be switched.

2.3 Clients: A Few Policies for Cache management

As we have discussed in section (2.1), tuning the broadcast program will eventually reach a limit due to the fixed bandwidth. Thus improving the broadcast for any one access probability will hurt the performance of clients with different access probability[1] The way breaking the upper bound of performance is to exploit the local memory and disk of the client machines to cache pages from the broadcast channel.

The clients are characterized by five parameters in the broadcast systems.

1. *CacheSize* – this is the number of pages that a client can store in its memory.

2. *ThinkTime* – this adjusts the requesting time of a client with the broadcast time unit of the server. It models the workload in the clients as well as the relative speeds of the CPUs of the client with respect to the server speed.

3. *AccessRange* – a client access to only a subset of the database, this reflects the nature of sharing in a broadcast system.

4. θ – the demanding frequencies for pages in the access range is modeled by a Zipf function $(i/N)^{\theta}$ exponential function. As shown in figure (4), the larger the θ is, the bigger the variations in the demanding frequencies.

[1]It seems not precise to say that this is a zero-sum game as mentioned in [1]. The performance has an upper bound as specified in the theorem in section (2.1). Tuning the broadcast probability in the wrong way may hurt all clients with no gain but loss.

5. *RegionSize* – This is similar to the bucket size in deciding the broadcast frequency. In the client, pages in the access range are also divided into finite buckets.

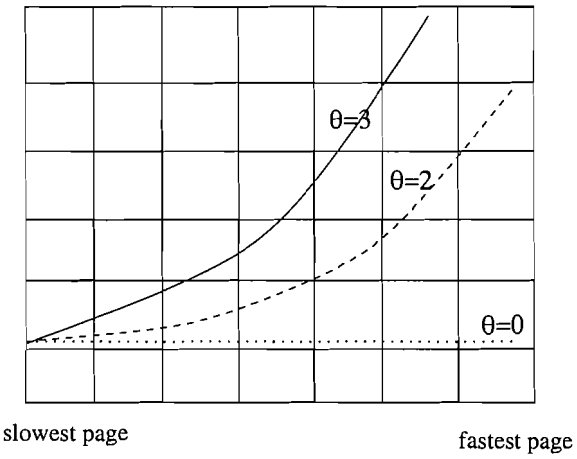


Figure 4 The access frequency of the pages is modeled by a Zipf function $F_i \propto i^{\theta}$. Thus θ controls the gradients of the frequency variation. The larger the θ is, the bigger variation the access frequencies have. When $\theta = 0$ it reduces to a flat pattern.

In the traditional pull-based caching management, the clients always cache their hottest pages. However, as we demonstrated in simulation that this cache policy leads to poor performance in the new push-based systems. In the broadcast disks, pages are pushed at various periods, and thus all non cache resident pages are not equidistant from the clients. Thus it makes more sense for the a client to store those *pages for which the local access probability is significantly greater than the page's frequency of broadcast.* This leads to the "cost-based" page replacement policy in [1].

In summary, the broadcast disks are designed in two separate steps: First, fixing the average access frequency, determine the broadcast frequency for the server. Second, given the broadcast page frequencies, determine the cache management policy for the clients. It can be designed in the inverse order. First, one chooses

a cache management policy, secondly one chooses the broadcast program. For example, when a P-policy is adopted in the client, then an offset is used in the broadcast disks.

In both cases, the system performance resulting from the two step design may not be optimal, but is still a very good solution in practice, given the current lack of theory in analyzing the cache management together with the broadcast frequency.

In the performance evaluation experiments, we choose to compare the following cache management policies, which are defined in the following.

1. **Least Recently Used (LRU)**. LRU maintain a linked list of all the pages in a cache. When a page is called, it is moved to the top of the list. Once a page is missing, the client waits for the page to appear in the broadcast channel. Then the new page is again put on the top of the list, and the page at the bottom of the list is chosen as victim for replacement.

2. **P-policy**. This policy simply saves the *CacheSize* pages whose access frequency are the highest at the client. Of course, this assumes that we know exactly the access frequency in advance, which may not be practical. The offset in the broadcast program is in fact designed for this P-policy. In that case, it is still a two step separate design – first the cache management, second the broadcast policy.

3. *PIX* **(P Inverse X)**. Suppose a page is broadcast at frequency P, and a client request this page at frequency X. Thus we assign a cost C for each page at the cache. Firstly, C should be proportional to P, because if a page is easy to capture from the channel (high P), then it is costly to store it in the cache. Secondly, if a page is requested at low frequency (small X), then it is also costly to store. Thus one can choose $C = P/X$. This is the PIX policy. Therefore suppose the broadcast frequency P and the access frequency X are

known in advance for all the pages. A client can simply compute C for all pages in its access range, and store the $CacheSize$ pages with least c. No page replacement is needed. It can be shown that the PIX policy is the optimal choice in the two-step designed systems.

4. *LIX*-**policy**. Neither P nor PIX are implementable in practice, as the page access frequencies are, in practice, unavailable. Thus they can only serve as ideal cases for comparison in simulations. The LIX policy is an approximation to the PIX by estimating the access frequency on-line, and it is derived from LRU. Unlike LRU which maintains a single link of pages for replacement, LIX maintains a $NumDisks$ short chains–each chain corresponds to a broadcast disk in the server side. However, the length of the short chains can vary over time, while their sum is fixed to be $CacheSize$. [2] The client keeps track of two values for each cache resident page. 1). $P_j.AccessProb$ initialized as zero, and 2). $p_j.LastAccessTime$ to record the last time of access. Thus when a page p_j is accessed, the client program updates its access probability estimation by the following formulas.

$$p_j.AccessProb = \alpha \frac{1}{CurrentTime \; - \; p_j.LastAccessTime} + (1-\alpha) p_j.AccessProb.$$

where $\alpha = 0.25$ is a decay factor. Then the program compute the cost $lix_j = P_j / p_j.AccessProb$ for the pages at the end of each list. where P_j is the broadcast frequency of page j. When a page is hit, it is moved to the top of the corresponding list, and the page whose lix_j is the smallest will be kicked out of the cache. Thus some lists may grow and some lists shrink dynamically in the process.

5. **L-policy**. L is exactly like LIX, except that it assumes that P_j being constant.

[2] Since a client's page range is only a subset of the database, a client may request no pages from some disks, then their corresponding cache lists have zero length.

6. **Prefetching**. Prefetch requests a page before it is actually accessed. Thus when a page is called, it may have resided in the cache. The advantage is obvious, as it reduces the waiting time. But as it uses up cache space, thus a trade-off has to be carefully considered between space and performance.

7. **PT: prefetch policy**. This is a simple prefetching method which is based on two measures: 1). p: the page access probability at this client, 2). t: the time elapsed before the next broadcast for the page. This is similar to some highway sign that reads: "Next gas station, 50 miles" or "next exit 40 miles", when the driver sees such warning signs, she/he will consider if he needs to add gas or take a break in the current exit. Intuitively, a page has larger value for prefetching if p or t is larger. Thus the PT policy compute $PT_i = p * t$ for each page i. When a page passes by, a PT_i is computed, and if it is larger than the PT_j for a page in the current cache, page i will replace j in the cache. Note that the PT value for a page is changing dynamically, as t is changing all the time.

8. **APT**. Like P and PIX, PT is impractical, as it needs to compute PT for all pages in the cache, which is too time consuming to sample the broadcast channel. Thus it can only be used as an optimal criterion for comparison. Instead, an approximative policy is proposed–the APT method. APT reduces the computational load by dividing the pages in the cache into blocks or regions, after sorting their probability of access. Thus it only considers the least important pages in each block as candidate of replacement. Therefore it reduces the computation to the order of $NumBlock$ in contrast to the $CacheSize$ in PT.

3 The Prototype System

This prototype system is mainly motivated by the following reasons:

- To validate the simulation results obtained by software simulations in previous publications[1, 2, 4]. It
- To test the feasibility of the push-based system using off the shelf hardwares.
- To test issues not addressed in the software simulation.

The prototype system is configured as displayed in figure (5). It uses four pentium based computers running Windows NT. One is used as server, and three are clients. The number of clients can be very large. The server was an Intel machine with 200 MHz pentium Pro processor and 96 MB of memory. The three clients had 200 MHz pentium processors with 48MB of memory. The computers are connected through a 10 or 100 Mbit/sec ethernet. IP multi-cast was used for data delivery from the server to the clients. The programming was written in the C++ language. The detailed set-ups of the prototype system is refereed to [6].

In the prototype system, the client runs a loop requesting pages using a skewed Zipf distribution, and the server broadcasts pages at the maximum rate possible. The actual rate depends on the bandwidth and the overhead of the CPU in running other programs, such as the operating system.

One main difference between the simulation and the prototype system is the measure of performance. In simulation[4], the response time was measured in broadcast units. The server broadcasts at a rate determined by the clients. In the prototype system, the server doesn't know the client processing capabilities. Thus a slow client may have to drop pages from the channel due to overflow of the buffers. The response time was also measured in real time (seconds) in the prototype system that reflects the exact time costs including overheads.

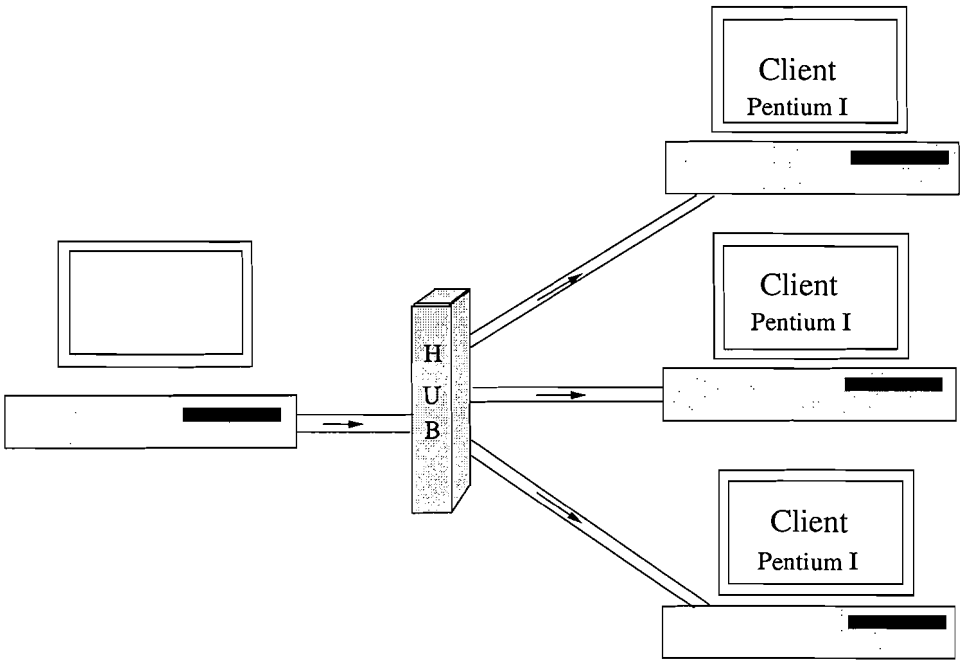


Figure 5 The prototype system with 4 PCs: one for server and three as clients.

4 Experiments

In this section, we briefly list some selected experiments. Since some of the other experiments were re-run by Rahul Bose and Swarup Acharya later on after adjusting some parameters, we choose not to discuss those experiments. For the experiment data below, they are mostly self-evident, and more detailed explanations are referred to Acharya's Ph.D thesis[5].

In general, the experiments on prototype system above faithfully reflect the conclusions drawn in the computer simulations, except that some implementation issues were not well foreseen in previous work[1]. Specifically, the results generated from the prototype system were found to be within a few percentage points of the previous simulation results. This testifies that the simulation results in [1] can be used to characterize a broadcast environment.

Result I: Comparison between PIX and P. The results are plot in figures (6) and

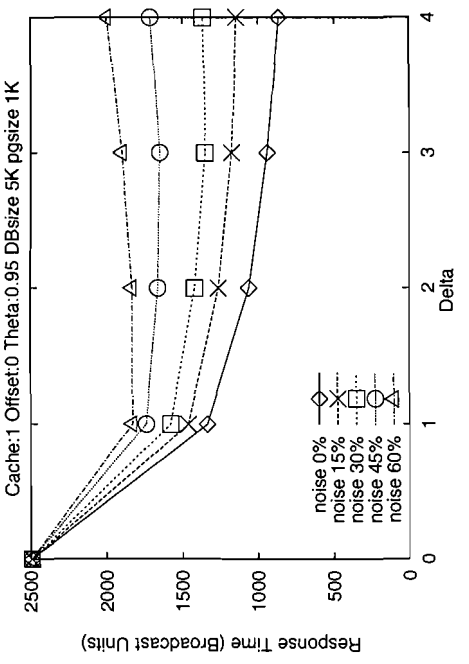
(7). In this experiment we compare the P and PIX policies with varying Delta and noise, we record the response time in both real time(seconds) and in broadcast unit. Figure (6.c) shows the actual bandwidth during the experiment which is almost fixed at 7.7 MBits. Figure (6.a & b) show that the response time are the same for both P and PIX when no cache is used on the client side. Figure (7) plots the response time for PIX and P when cache size is 500 for the client. PIX outperforms P when Delta is not zero, i.e. the Broadcast disks beats the flat disk.

Result II. The effects of cache sizes. The results for PT and PIX are plotted in figure (8). Since PT needs computational overhead in evaluating the priorities of each page in the cache, therefore the larger the cache, the longer time it costs. As a result, more pages is dropped as the cache size increases, see figure (8.c). Since PT maintains a better cache, it has a short response time even it drops more pages, see figure (8.a). PIX has a better cache hit rate than PT when cache size is small, and has a worse cache hit rate when cache size is large, see figure (8.d).

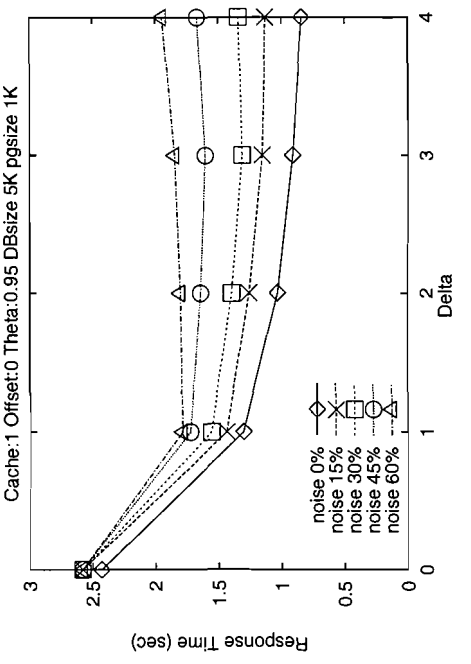
Result III. The effects of *Delta*. The results are plotted in figures (9) and (10). Cache hit rate keeps the same while Delta increases, see figure (9.c). Response time decreases while Delta increases, see figure (9.c) and figure (10). Figure (9.a) shows the page dropping rates fluctuate between 0 to 0.011, which are very close to zero.

Result IV. The effects of *noise*. The results are plotted in figures (11) and (12). Cache hit rates slightly decrease while noise increases, see figure (11.c). Dropped pages increase while noise increases, see figure (11.a). Response time increases while noise increases, see figure (11.d) & figure (12).

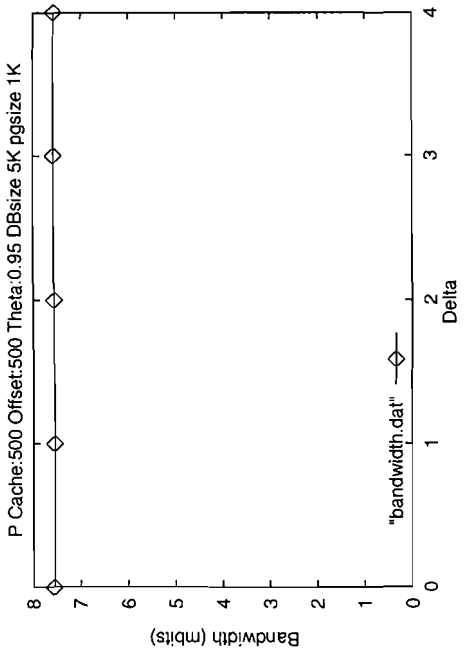
Result V. Comparing PIX vs. PT vs. APT with noise changes. The results are plotted in figures (13) and (14). PIX has the best cache hit rate, APT has the worst cache hit rate regardless to the noise, see figure (13.c). PIX has the worst response time, PT has the best response time regardless of noise, see figure (13.d)



a



b



c

Figure 6 PIX vs. P

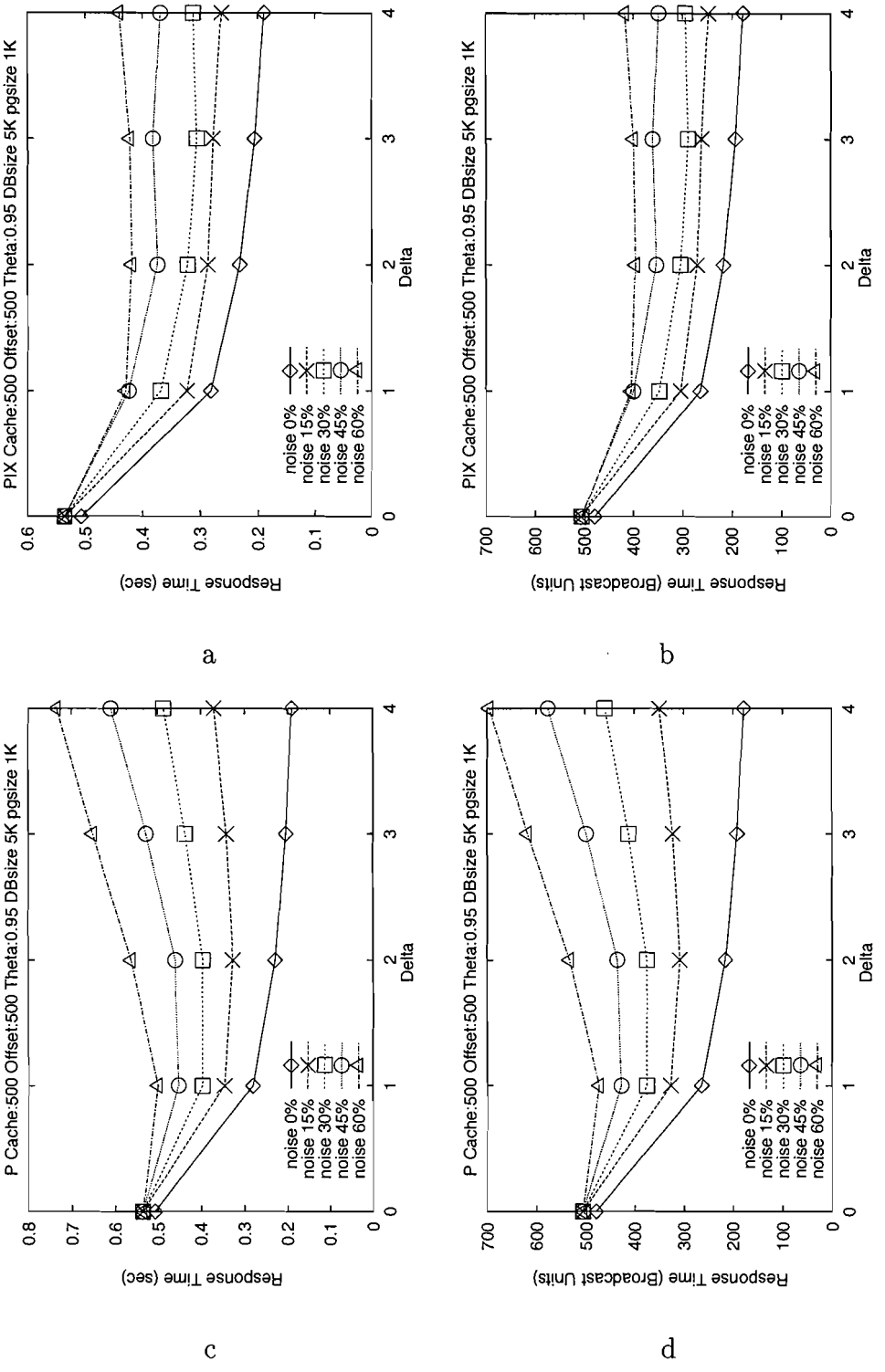


Figure 7 PIX vs. P (continue)

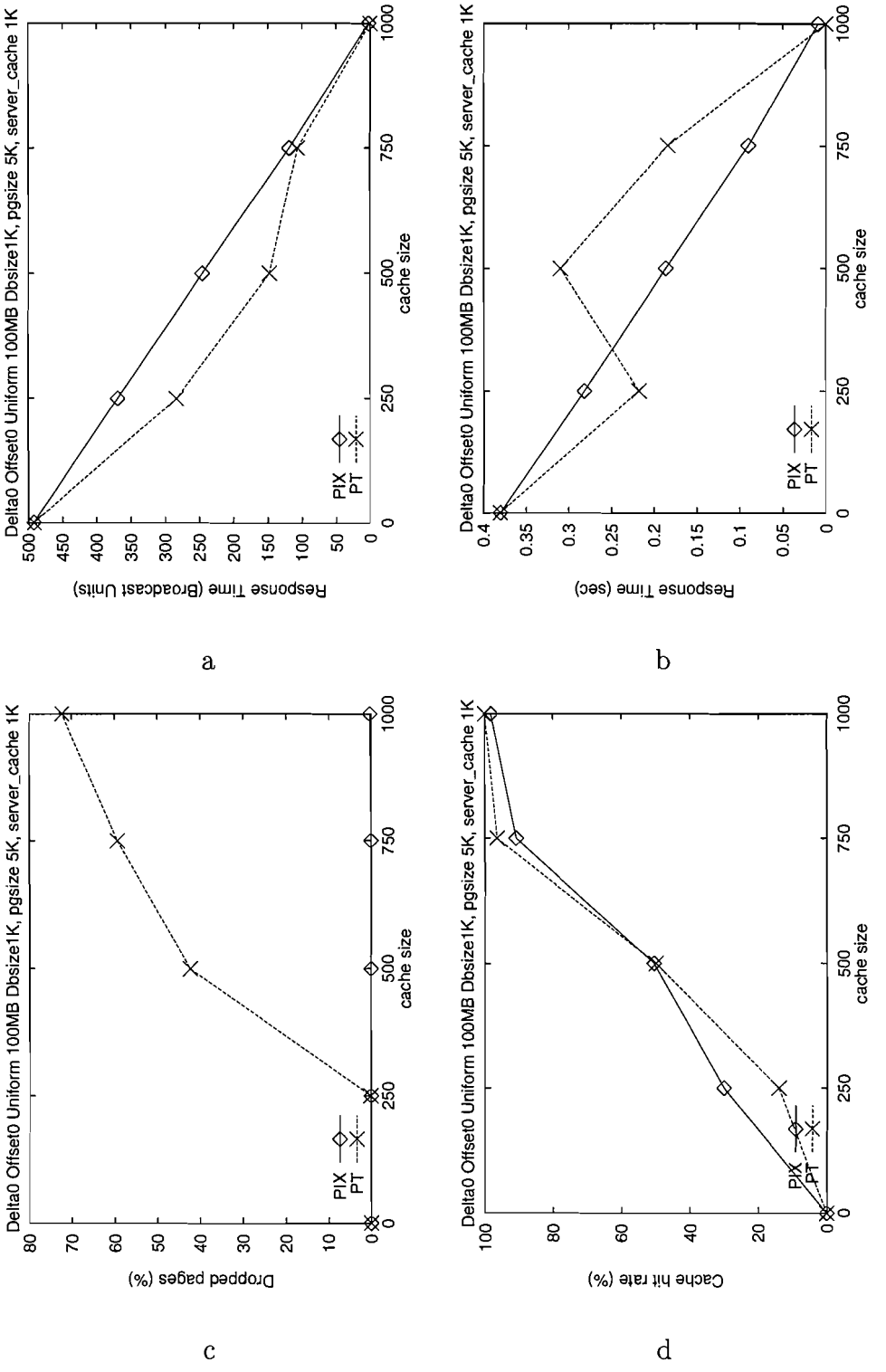


Figure 8 The effects of cache sizes.

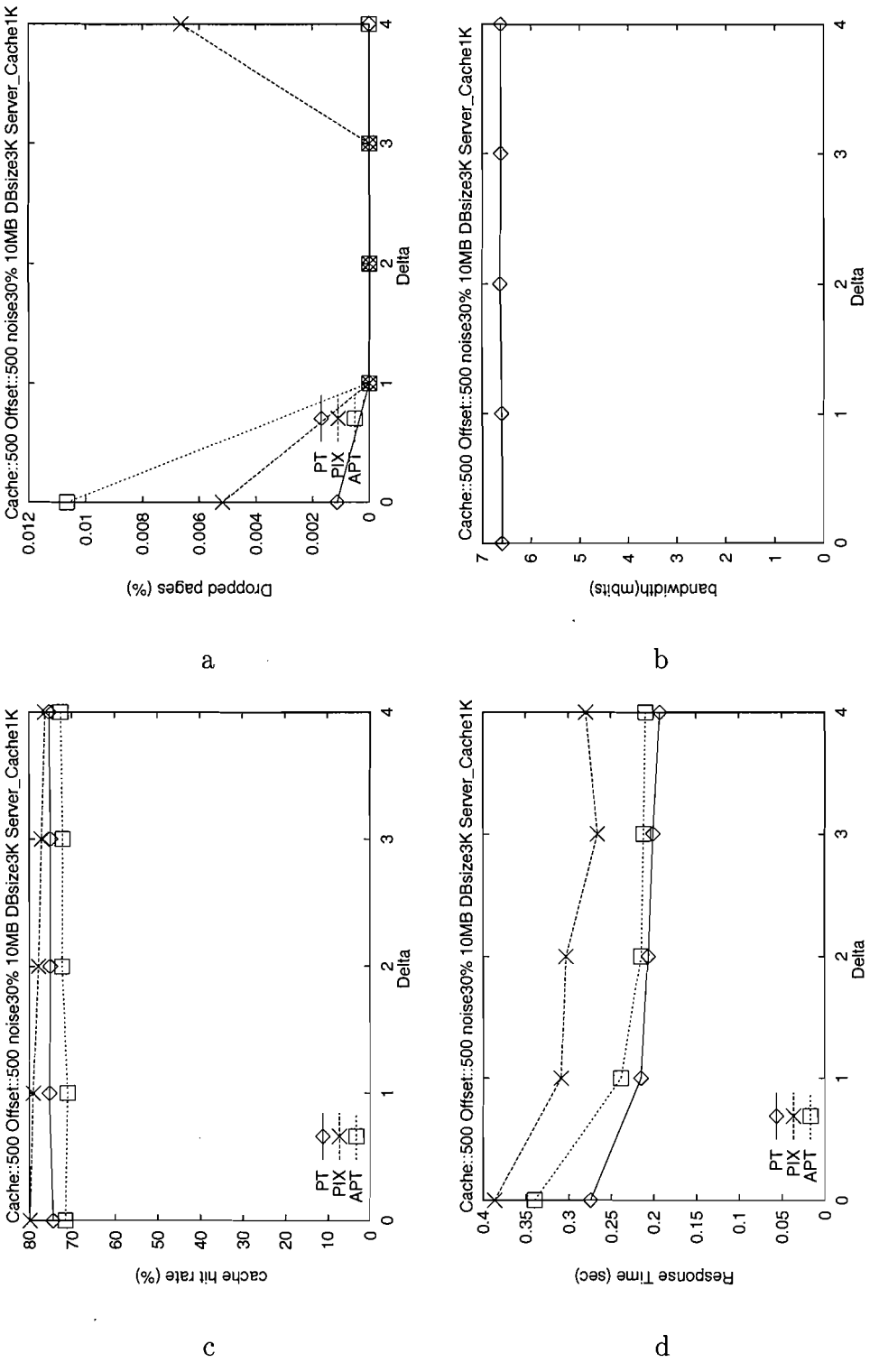


Figure 9 The effects of *Delta*.

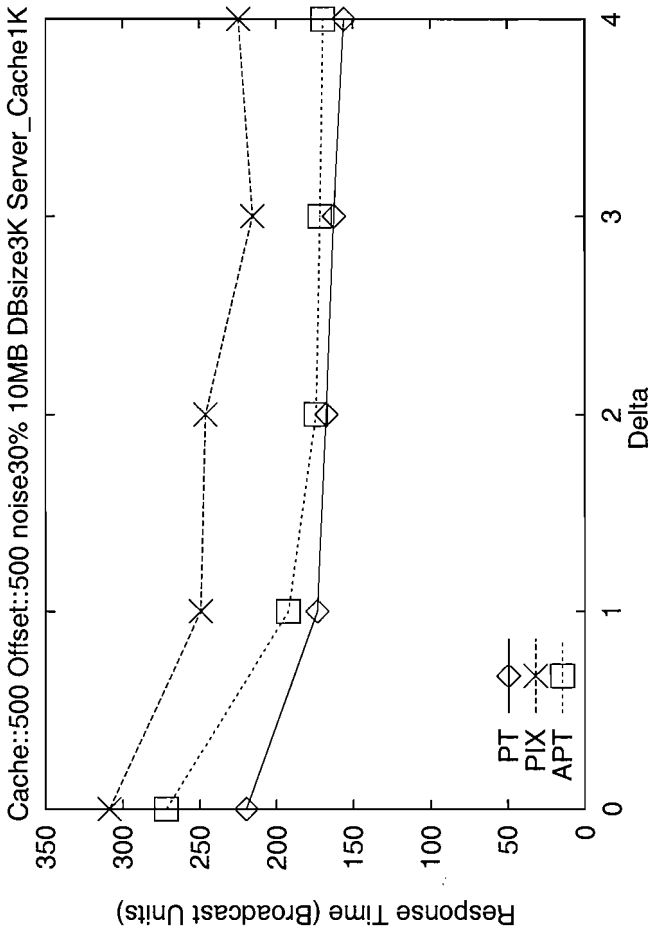


Figure 10 The effects of *Delta* (continue).

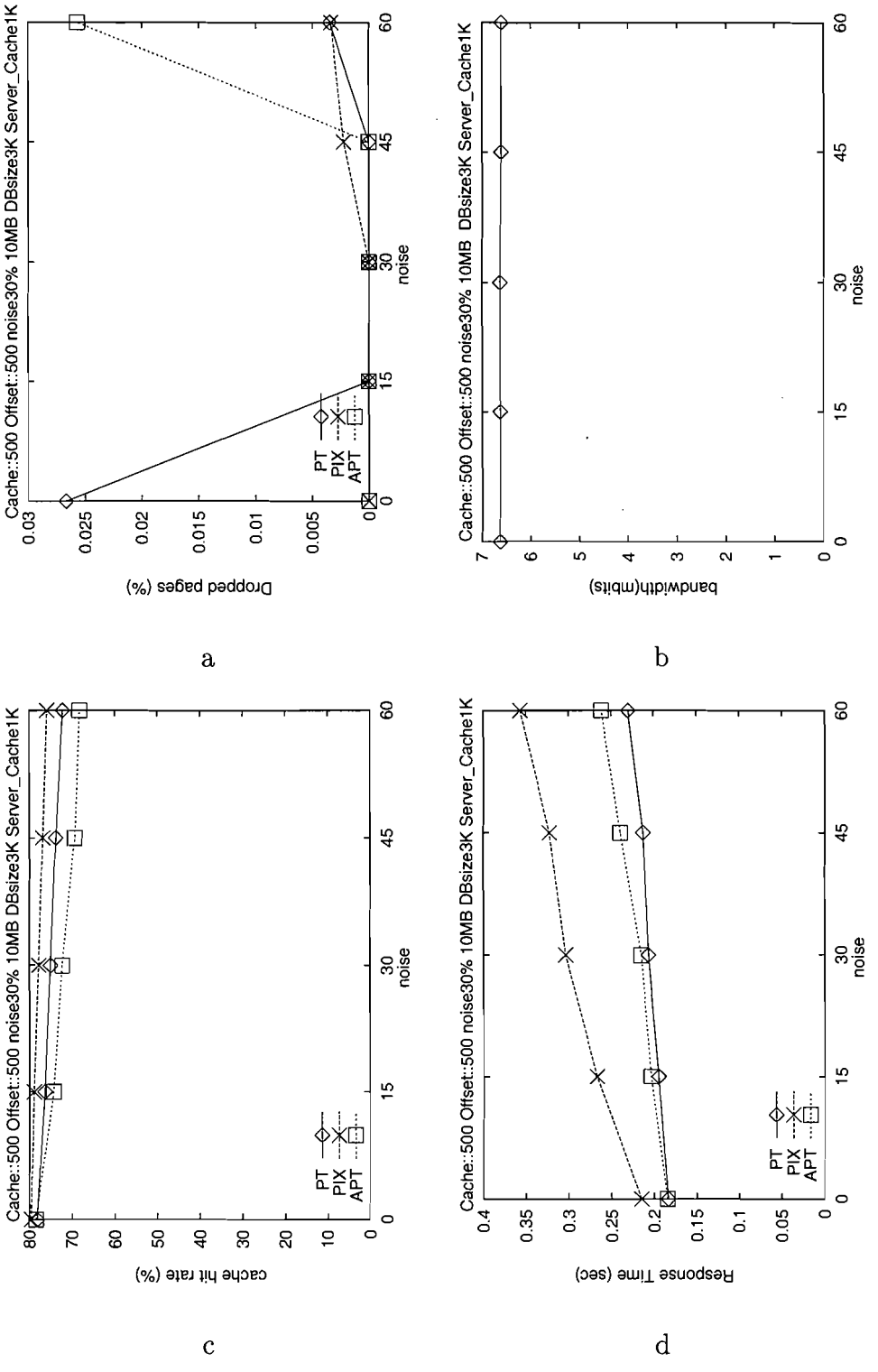


Figure 11 The effects of *noise*.

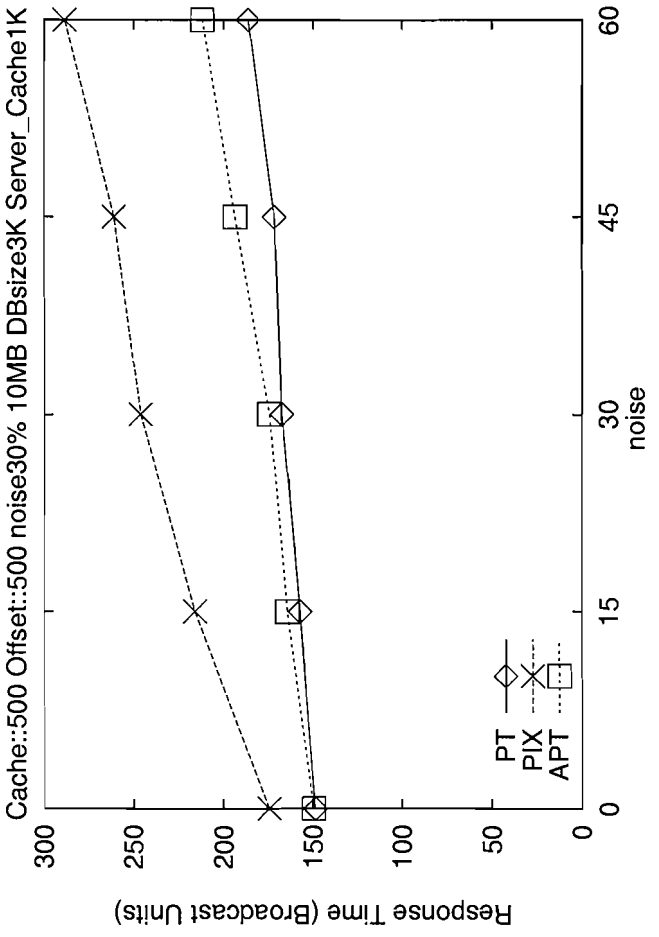


Figure 12 The effects of *noise* (continue).

& figure (14). APT drops the most pages and PT drops least pages regardless of noise, see figure (13.a).

Result VI. Comparing PIX vs. PT vs. APT with Δ changes. The results are plotted in figures (15) and (16). PIX has the best cache hit rate while APT has the worst cache hit rate regardless of *Delta*, see figure (15.c). PIX has the worst response time while PT has the best response time regardless of Δ, see figure (15.d) & figure (16). APT drops the most pages and PT drops least pages regardless of Δ, see figure (15.a). PT takes the time to maintain a better cache, drops more pages, gets the best performance. APT drops the most pages, still outperforms the PIX.

5 Conclusions

Through the work of this thesis, I have learnt the broadcast disks system, and did experiments on a prototype system. The research is of great values to my professional career. During the work, I also learn to formulate a problem using mathematical tool, and learn to compare various parameters and management policies carefully in order to draw firm conclusions.

Technically, the performance of prototype system confirms the following conclusions.

1. Firstly, multiple broadcast disks in the push-based system is clearly better for the skewed page access frequency.

2. Secondly, the push-based architecture profoundly change the cache management concept in traditional pull-based systems. This was confirmed by the results that PIX/LIX out-performs the traditional LRU policy.

3. Thirdly, prefetching indeed improves the performance. However, if the clients fails to keep up with the broadcast speed (page drops off due to extra load

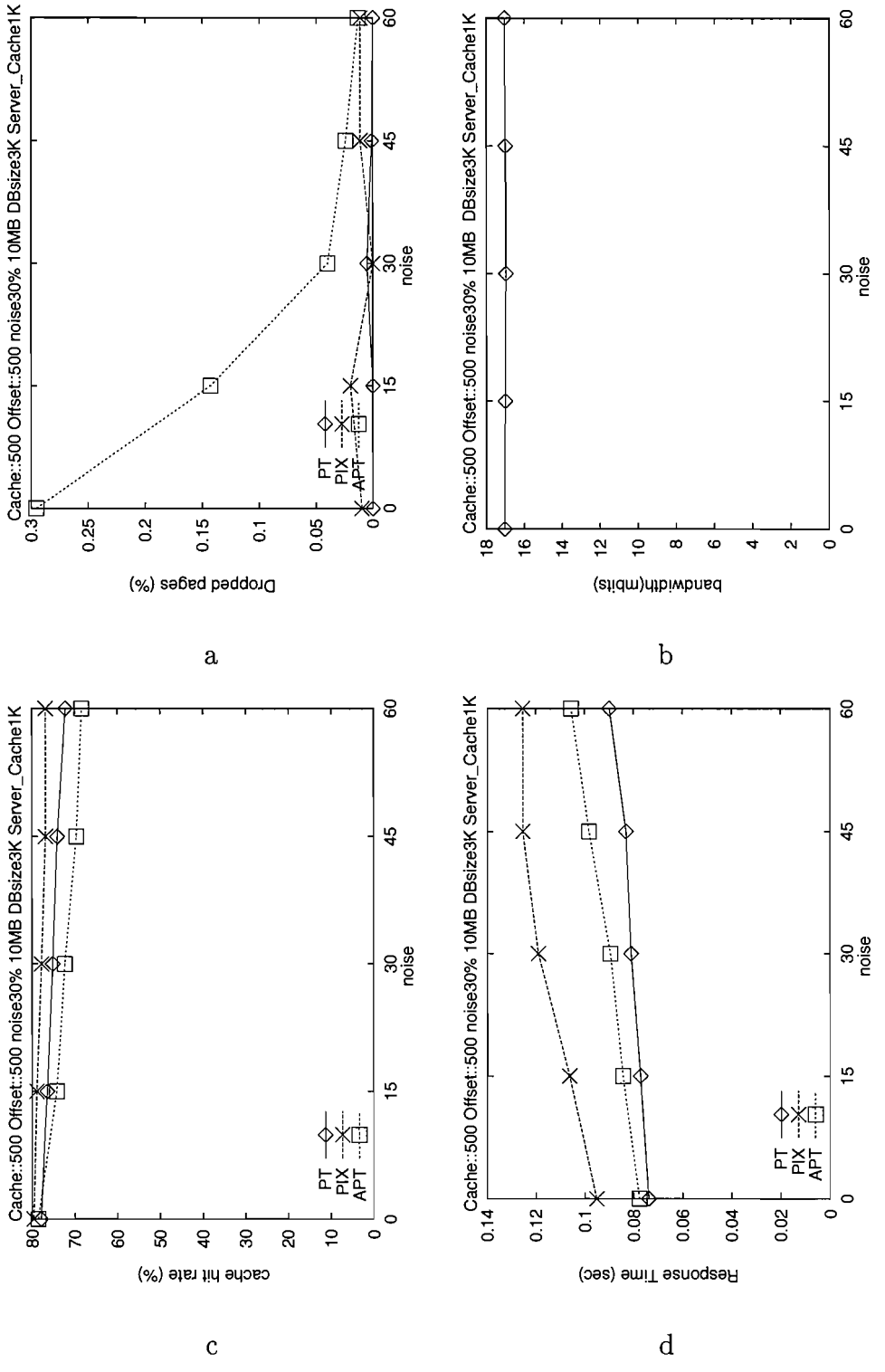


Figure 13 The effects of *noise* sizes.

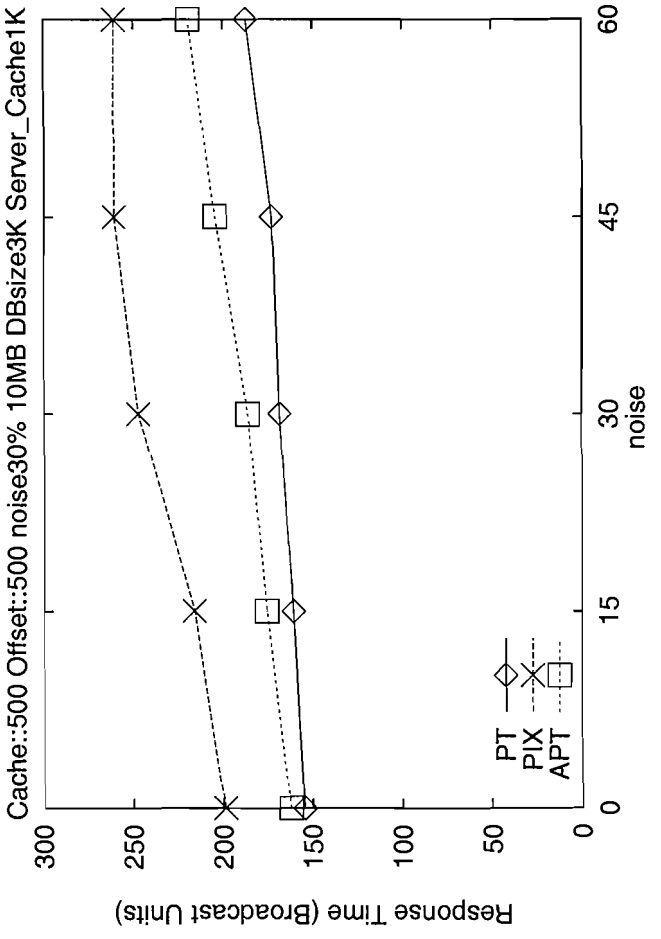


Figure 14 The effects of *noise* sizes (continue).

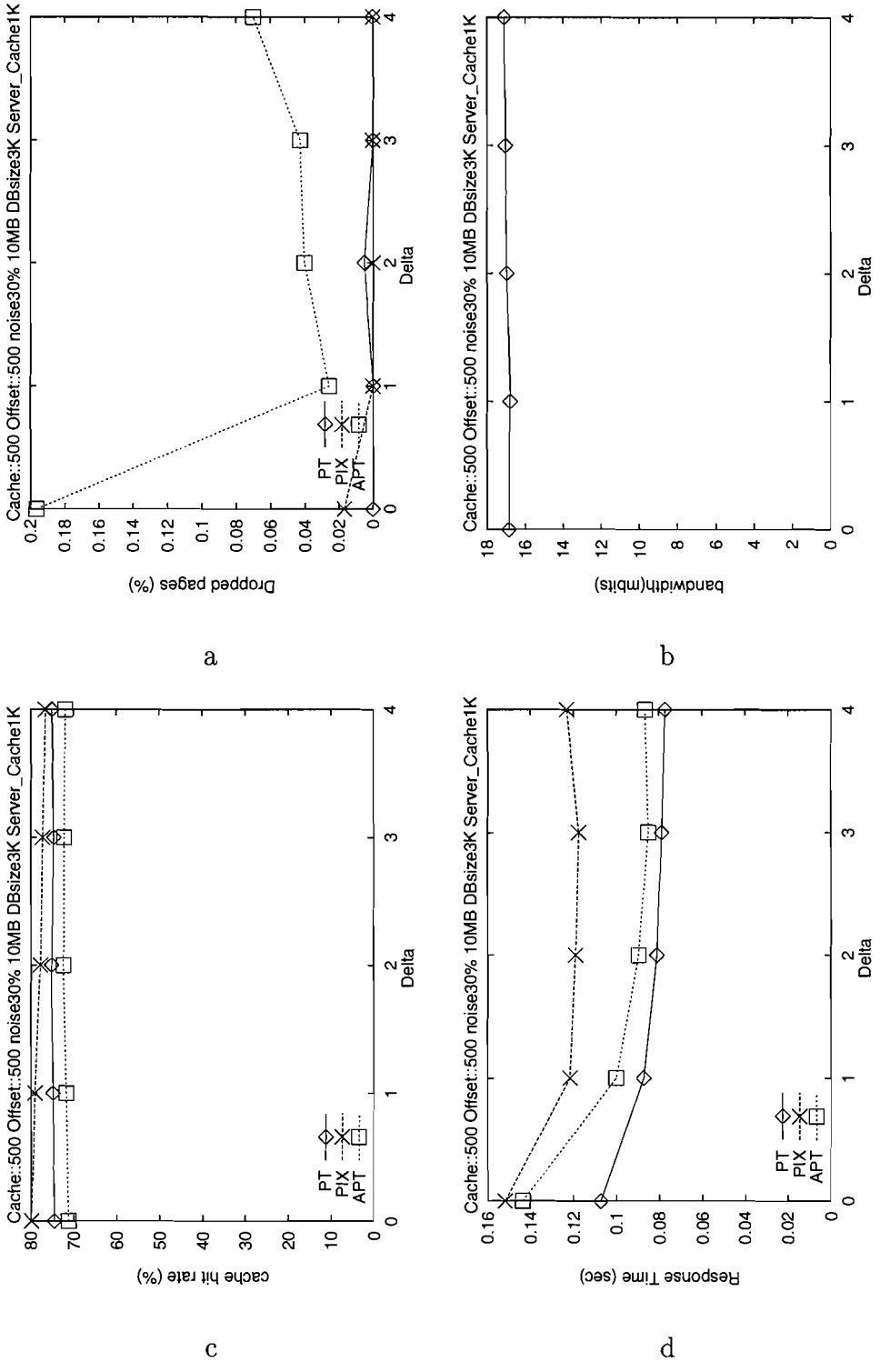


Figure 15 The effects of *Delta* sizes.

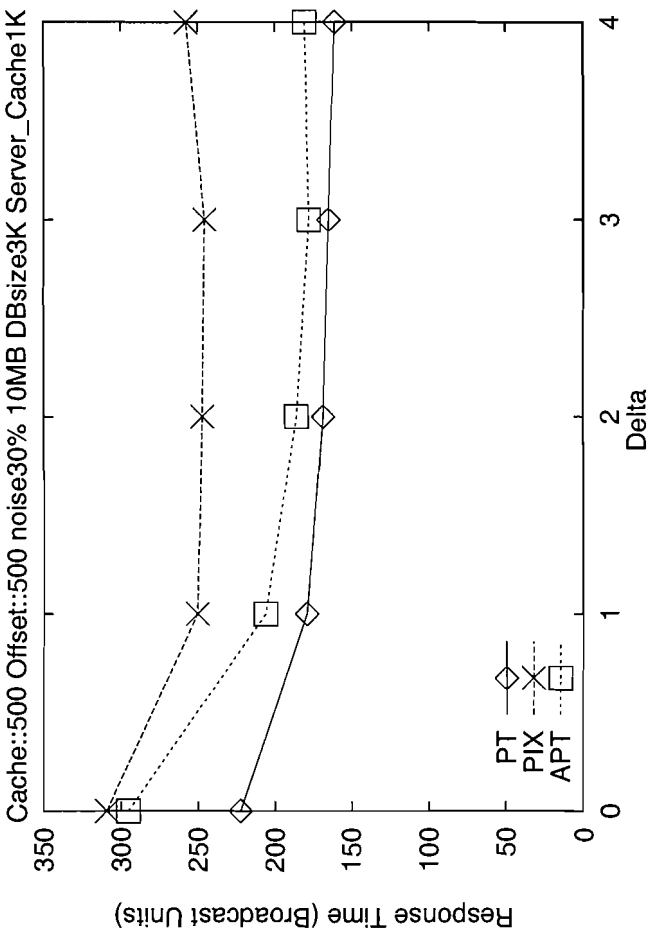


Figure 16 The effects of *Delta* sizes.

for prefetching calculation), then the prefetching method PT was found to degraded below PIX in performance.

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: data management for asymmetric communication environments", *Int'l Conf. on Management of Data*, pp 199-210, CA, 1995.

[2] S. Acharya, M. Franklin, and S. Zdonik, "Disseminating updates on broadcast disks", *Proc. of 22nd VLDB*, september, 1996.

[3] S. Acharya, M. Franklin, and S. Zdonik, "Prefetching from a broadcast disk", *Proc. 12th Int'l Conf. on Data Engineering*, Feb. 1996.

[4] S. Acharya, M. Franklin, and S. Zdonik, "Balancing push and pull for data broadcast", *Proc. ACM SIGMOD*, May, 1997.

[5] S. Acharya, *Broadcast Disks: Dissemination-based Data Management for Asymmetric Communication Environments*. Ph.D Dissertation, Computer Science Department, Brown University, 1998.

[6] R. Bose, *Cache management in push-based systems*, Master's Thesis, Brown University, Providence, RI, 02912, Oct 1997.

[7] T. Bowen, G. Gopal, G. Herman, etc. "The Datacycle Architecture", *CACM 35*, (12), 1992.

[8] M. Carey, M. Franklin, M. Livny, E. Shekita, "Data caching tradeoffs in the client-server DBMS architectures", *Proc. ACM SIGMOD conf.* , Denver, 1991.

[9] M. Franklin and M. Carey, "Client-server caching revisited", *Proc. Int'l workshop on Distributed Object Management*, August, 1992.

[10] D. Knuth, *The Art of Computer Programming*, Addison Wesley, 1981.

[11] Y. Wang and L. Rowe, "Cache consistency and concurrent control in a client/server DBMS architecture", *Proc. ACM SIGMOD Conf.* Denver, June, 1991.

[12] S. Zdonik, M. Franklin, R. Alonso, and S. Acharya, "Are disks in the air just pie in the sky?", *IEEE workshop on Mobile Computing Systems and Applications*, CA, Dec. 1994.