Adaptive Data Broadcast in Hybrid Networks

by K. Stathatos, N. Roussopoulos, J.S. Baras

CSHCN T.R. 97-11
(ISR T.R. 97-40)

Adaptive Data Broadcast in Hybrid Networks[*]

Konstantinos Stathatos[†] Nick Roussopoulos[†] John S. Baras[‡]

kostas@cs.umd.edu nick@cs.umd.edu baras@isr.umd.edu

Center for Satellite and Hybrid Communication Networks

Institute for Systems Research

University of Maryland, College Park, MD, 20742

April 8, 1997

Abstract

Hybrid networks combine multiple communication modes and are fast emerging as the most viable solution for the ever increasing demand for bandwidth and data services. Taking advantage of this new technology, we are proposing a hybrid scheme which effectively combines broadcast for massive data dissemination and unicast for individual data delivery. The goal is to build highly scalable systems with small response time. In this paper, we describe a technique that continuously adapts the broadcast content to match the hot-spot of the workload. We show that the hot-spot can be accurately obtained by monitoring the "broadcast misses" observed through direct requests. This is a major departure from all other broadcast optimization schemes which are handicapped by their total reliance on complete knowledge of both "hits" and "misses". We also show that the proposed adaptive scheme performs effectively even under very dynamic and rapidly changing workloads. Extensive simulation results demonstrate both the scalability and versatility of the technique. Another basic result obtained in this paper is that the overall system's throughput depends only on the size of the hot-spot and not on the volume of the workload. This has far reaching implications for very large scale and high volume wide area information systems.

[*]This material is based upon work supported by the Center for Satellite and Hybrid Communication Networks under NASA grant NAGW–2777, by the National Science Foundation under Grants No. NSF EEC 94–02384 and No. ASC 9318183, and by ARPA under Grant No. F30602–93–C–0177

[†]Also with the Dept. of Computer Science

[‡]Also with the Dept. of Electrical Engineering

1 Introduction

The world is witnessing an unprecedented demand for data services. The immense popularity of the Web has generated exponential demand workloads that cannot be satisfied with existing Internet capacity and traditional data services in which scalability grows at best linearly with network bandwidth and server capacity. Such workloads are observed in the course of special events such as the Atlanta Olympics, the presidential elections, etc. when several million requests per minute are made during peak periods.

Traditional unicast (point-to-point) and connection-oriented data services are uneconomical because even if the infrastructure were developed to meet the demand in both network bandwidth and server capacity, most of this infrastructure would be underutilized and wasted during non-peak periods. Data broadcast, on the other hand, exploits hybrid network architectures to asynchronously deliver data in a connection-less mode and has the advantage of meeting such workloads provided that the "hot-set" can be identified and delivered through the broadcast channels. The data is delivered to all clients simultaneously and, therefore, the number of clients is unlimited while the quality of service (i.e. response time) remains the same. In fact, the more the simultaneous clients, the more economical the cost of such data services.

Clearly, the key issue is the identification of the "hot-set" for connection-less delivery through broadcast while serving only "broadcast misses" in the usual connection-oriented way. However, there are two major obstacles. First, data needs can be neither characterized nor predicted a-priori because of the dynamic nature of the demand. Special events which produce explosive access workloads such as emergency or weather related situations cannot be precompiled into optimal schedules. Thus, techniques based on precompiled schedules [AAFZ95, ST97] are not applicable in this case. Second, passive broadcast systems are limited because clients have no means of communicating neither their data needs, which can dynamically change, nor how useful the content of the broadcast is. In other words, neither hits nor misses on the content are reported.

In [SRB96] we introduced the idea of a hybrid scheme which effectively combines broadcast for massive data dissemination and unicast for upon-request data delivery. The goal is to build highly scalable systems with small response time. In this paper, we propose a technique that achieves that by continuously adapting the broadcast content to match the hot-spot of the workload. We show that the hot-spot can be accurately obtained by monitoring the "broadcast misses" observed through direct requests. This is a major departure from all other broadcast optimization schemes which are handicapped by their total reliance on complete knowledge of both "hits" and "misses". We develop an adaptive algorithm which relies on "marginal gains" and "probing" to identify the more intensively requested data. We show that the overall performance of this hybrid system can surpass the capacity of the data servers by several orders of magnitude under the following two assumptions: a) the size of hot spot and the network bandwidth are such that the hot-spot can be broadcast with acceptable response time, and b) the remaining cold set is sustained below the data server's capacity. If these two assumptions hold, the performance of the hybrid system proposed herein is independent of the total volume of the workload and, thus, the system exhibits unlimited scalability.

Section 2 presents a basic analytical model for hybrid data delivery and discusses the type of workloads that can benefit from it. Then, in section 3 we describe the adaptive algorithm and its refinements. Section 4 describes the simulation model we implemented to evaluate our system as well as the experiments we performed. Finally, we refer to related work, in section 5, and conclude with a summary of our results in section 6.

2 Balancing Data Delivery

2.1 The Hybrid Model

In this section, we develop a simplified analytical model for the hybrid delivery scheme which will provide the some intuition behind our work and illustrate the involved trade-offs. Let us consider a system that broadcasts data with period T over a channel of bandwidth B. Often, this periodic broadcast is perceived as a special memory space (data store) of the server of size $B \times T$. Its major advantage is that it can be accessed by any number of clients concurrently (i.e. no access contention) with no overhead for the server. Its only limitation is that it can be accessed only sequentially since clients need to wait for the data of their interest to appear on the channel. A direct consequence is that the average access time depends on the size of that memory which in turn is determined by the period T.

In modern hybrid/overlay communication environments, we can exploit multiple data delivery paths. Broadcast and unicast, possibly through different communication channels, can be combined to work synergistically and yield higher data delivery rates. For exploiting the characteristics of each of the data paths, we should be looking for solutions that range between:

Broadcast everything (pure data push): This scheme includes the case when clients are completely passive listeners who make no explicit requests. Such a scheme can accommodate an arbitrary large number of clients (unlimited scalability). But, the average data access time grows with the size of the database which has to be fairly small to keep the response time acceptable.

Broadcast nothing (pure data pull): All requests are explicitly made to the server, and thus this becomes just a standard client server architecture. Such a scheme cannot scale beyond the server's and the network's maximum capacity. The average data access time depends on the aggregate server workload as well as the network load, but not on the size of the database.

Consider a database containing N data items of equal size S. The demand for each item i forms a Poisson process of rate λ_i. Assume that the data items are numbered such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N$. A server[1] can service requests for these items with mean service time $1/\mu$. In addition, this server can broadcast data over a channel at a rate B. Assume that, for some reason, the server decides to broadcast (push) the n first items and offer the rest on-demand (when pulled). If we define $\Lambda_k = \sum_{i=1}^{k} \lambda_i$, the overall expected response time of the system is

$$T = \frac{\Lambda_n T_{push} + (\Lambda_N - \Lambda_n) T_{pull}}{\Lambda_N}$$

where $T_{pull} = \frac{1}{\mu - (\Lambda_N - \Lambda_n)}$ is the expected response time for on-demand requests and $T_{push} = \frac{n\,S}{2B}$ is that of the requests satisfied by the broadcast (equal to half of the broadcast period). Figure 1 plots a representative example of the response times T, T_{pull} and T_{push} as a function of the number n of broadcast items. We have assumed that the total workload, represented by Λ_N, is greater than μ, which is a safe assumption for large scale systems with huge client populations.

The first thing to note in figure 1, is that the performance of the data-pull service is exponentially affected by the imposed load. It is evident that with too little broadcasting, the volume of requests at the server may increase beyond its capacity, making service practically impossible (left side of the graph). Stated more formally, the response time for pulled data—and consequently the overall response time—grows arbitrarily large, for $n < M$, where M is such that $\Lambda_N - \Lambda_M = \mu$. On the other hand, the response time for pushed data is a straight line, growing proportionally

[1]For simplicity we model the server as an M/M/1 system

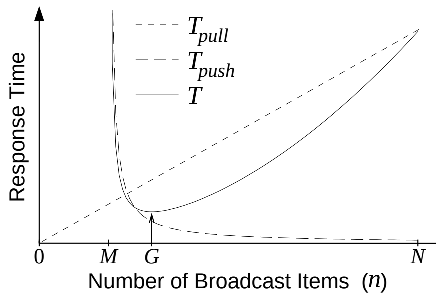

Figure 1: Hybrid Delivery Trade-off

with the size of the broadcast[2]. The slope of that line is determined by the size of the data S and the available bandwidth B. As a result, too much broadcasting is not desirable either, and in order to achieve best performance, we must look for solutions in the area around G, where we can maintain a proper balance between data push and pull.

2.2 Practical Considerations for Workloads

The discussion of the previous sections suggests that there is a way of balancing data delivery modes in a way optimal with respect to the overall response time. However, it is not all that obvious how good that optimal solution is, since in some cases this optimal response time is unacceptable. In what follows, we explore hybrid delivery from a practical perspective and give a qualitative answer to when the combination of broadcasting and unicasting can be advantageous.

Intuitively, data broadcasting is helpful when its content is useful to multiple receivers. If that happens the benefit is two fold: first, for each single broadcast message the server saves several unicast messages that otherwise would have to send individually, and second, the satisfied receivers avoid sending requests that might end up clogging the server. On the other hand, broadcast data that are useful to one or even no receiver do not yield any benefit, but instead, they harm overall performance since they occupy valuable bandwidth. This implies that we can use data broadcasting more effectively when there is some significant commonality of reference among the client population. Ideally, we would like to be able to detect and exploit that commonality.

Consider, for example, a data set of N items and assume that they get requested according to the skewed access pattern of figure 2. For clarity, we assume that items are sorted according to their respective request rates. From the discussion so far, it is clear that we are looking for the optimal point G to draw the line between data that should be pushed and data that are left to be

[2]This is a good approximation, although it may not be always true.

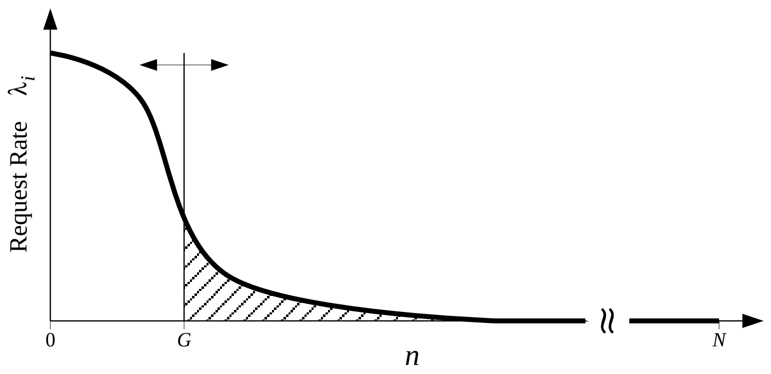

Figure 2: Skewed Data Access Pattern

pulled. The area to the left of G (the head of the distribution) represents the volume of requests satisfied by the broadcast. Note that only the width of this area affects the system's performance and not its height. In fact, the higher this area gets, the greater the benefit of broadcasting. The shaded area to the right of G (the tail of the distribution) represents the volume of the explicit requests directed to the server. Having this in mind, the selection of G should satisfy two constraints:

- the head should be wide enough to accommodate the hot-set but not so wide that it could yield unacceptable response time.
- the tail of the workload (shaded area) should be maintained below the server's capacity.

In practice, the shape of the tail is what determines the viability of the hybrid scheme. To make this clear, consider a case where the tail is a very long area of very small (but not zero) height. That would correspond to a very large number of items that get requested very infrequently. If the total tail area is larger than the server's capacity, then we need to move the point G even more to the right. But, since each item contributes very little to the total area, the optimal G would be found deep into this tail. This means that a lot of rarely requested items would be broadcast. In effect, that is a lose-lose situation because the quality and the performance of the broadcast is significantly reduced with minimum or no returned benefit. In such situations, the only solution is to increase the server's capacity to cover the tail.

Having this in mind, we consider cases where the optimal solution does not require broadcasting of rarely requested data. Under the assumption that the server's capacity is at least such that it can handle the aggregate load imposed by requests for such data, we propose an adaptive hybrid scheme which, in a near optimal way, exploits broadcasting to take the load of hot data off the server which is left with a tolerable load imposed by infrequently requested data.

3 Adaptive Hybrid Delivery

In this section we elaborate on the proposed adaptive hybrid delivery scheme. Our solution is mainly based on the notion of data caching. Conceptually, we treat the available broadcast capacity as a global cache memory between the server and the clients. Like typical cache memories, this "air-cache" should allow clients to get the data they need faster than directly from the server and at the same time limit contention for the shared resources (i.e. the server and the network). Similarly to caching in other contexts, the air-cache should be adaptive to the system's workload. The challenge in making it adaptive lies on the fact that the server cannot have a clear picture about the actual usage of the broadcast data simply because satisfied clients do not acknowledge the usefulness of the received data. Therefore, traditional cache management techniques, such as LRU, MRU, etc., are not applicable. However, "air-cache misses", indicated by explicit requests for data not broadcasted, provide the server with tangible statistics on their demand frequency. This unveils an interesting perplexity of the system: the more misses the better server statistics to adapt on. But, on the other hand, the more passive the clients are, the more satisfied they are with the broadcast.

3.1 Vapor, Liquid and Frigid Data

To facilitate the presentation of our work, we define three possible states for data and borrow terminology from their analogy to the three physical states of water. More specifically, for each item in the database we define a *temperature* which corresponds to its request rate λ_i. Based on their temperature, data items can be in one of three states:

Vapor (Steamy) Hot: Items for which the server has determined that are currently heavily requested and therefore should be broadcast, i.e. put in the air-cache.

Liquid Warm: Items currently not broadcast for which the server has recently received a moderate or small (not enough to justify broadcasting) number of requests.

Frigid (Icy) Cold: Items that have not been requested for a while and, therefore, their temperature λ_i has practically dropped to 0 (degrees centigrade).

In the adaptive scheme that we propose, the server dynamically determines the state of the database items based on the air-cache misses. These can be considered as the "sparks" that regulate the temperature and state of the data. They operate according to the following rules:

- For vapor data, we assume that clients always retrieve them from the broadcast channel[3], so there is no real feedback to the server about their actual temperature. In other words, there are no heated by requests and they gradually cool down until eventually they turn into liquid. The duration of this cooling process depends on the temperature that caused them to vaporize in the first place (the hotter they were, the longer it takes for them to cool down).

- Liquid data items that continue being requested either turn into vapor or remain liquid, depending on the intensity of the requests. If they stop being requested they eventually freeze.

- Frigid data items that start being requested turn into liquid or even vapor depending again on the intensity of their requests. Obviously, as long as there are no requests they remain frigid.

The hardest part of this process is in distinguishing vapor from liquid data, which is the focus of the rest of the paper. The distinction between liquid and frigid data items is, in effect, similar to that implicitly achieved by a buffer manager of a database system using a standard LRU replacement policy. Likewise, the server should tend to maintain liquid items in main memory anticipating new requests in the near future, and can retrieve frigid items from secondary or tertiary memory only when necessary. However, in practice the distinction of frigid data plays another important role in terms of overhead, specially in the common case where frigid data make up the largest part of the database. With a default 0 temperature, the server is off-loaded from keeping statistics for frigid data, and can also safely ignore them when looking for candidate vapor items.

3.2 Repetitive Data Broadcasting

Contrary to typical periodic broadcast schemes that require a fixed schedule, we employ a non–periodic repetitive broadcast scheme. Vapor data are repeatedly broadcast to achieve the effects

[3]We are also investigating the effects of relaxing this constraint

of caching in the broadcast channel, but the content of the broadcast is continuously updated to better match the workload. The heart of this approach is a queue $\mathcal{V}$ which the server uses to maintain all vapor data. The server picks the next item to broadcast from the head of $\mathcal{V}$. After it gets broadcast, the server removes it from the head and moves it back to the tail of $\mathcal{V}$. At the same time, its temperature is lowered by some predetermined factor to reflect the cooling process of vapor data.

In order to be able to update the contents of the queue, we assign one of the items in $\mathcal{V}$ as a placeholder. Once this placeholder is broadcast, the server re–evaluates the state of data and modifies the contents of the queue accordingly. This evaluation process, which is described in detail in the next section, selects which of the vapor items (if any) should be demoted to liquid and which of the liquid ones have heated up enough to be vaporized. The vapor items that are to be demoted are marked so that after their next broadcast they will be removed from the queue. The new vapor items are placed on the tail of queue so that they will start being broadcast. After this update, the final item on the tail of $\mathcal{V}$ is assigned as the new placeholder. The result of this technique is a repetitive scheme with continuously evolving content and no fixed period.

Another integral part of the hybrid delivery scheme is the indexing of the broadcast channel. Since the clients are expected to select between the two data delivery paths, the server needs to have a way of letting them know what to expect from the broadcast. For this, we have adopted a simple technique that uses the queue's signature (i.e. the list of data identifiers in the queue) as an index which is broadcast properly interleaved with the data. The clients examine the index and decide whether to wait for the required item to arrive or to make an explicit request for it. The frequency of index inclusion in the broadcast is determined based on the size of the data and the maximum time clients are willing to wait before making the decision. In the future we are planning to incorporate additional and more elaborate indexing schemes that take into account other trade-offs, such as those proposed in [IVB94b] and [IVB94a].

3.3 Adaptation Based on Marginal Gains

As mentioned in the previous sections, the server needs to dynamically determine the state of the various data items. The separation of liquid and frigid items is simply determined by the age of the last request for each item. The characterization of an item either as vapor or liquid is a lot more intricate. Every time the server has to re-evaluate the state of data, it actually needs to make two kinds of decisions: (a) which (if any) of the vapor data have cooled-down enough to be demoted to liquid, and (b) which of the liquid data became hot enough to be promoted to vapor. A straightforward approach of establishing absolute temperature thresholds for vapor and liquid data does not work because the state of the data depends also on the aggregate workload[4] which is dynamic and, in many cases, very bursty. In order to take into account the aggregate workload, we developed an algorithm that relies on the expected marginal gain or loss resulting from each possible decision.

T	:	average overall response time
T_{push}	:	average access time for pushed data
T_{pull}	:	average access time for pulled data
Λ_L	:	aggregate request rate for liquid data
Λ_V	:	aggregate request rate for vapor data
N_V	:	number of vapor items
S_i	:	size of the item i
λ_i	:	request rate (temperature) of the item i
μ	:	mean service rate of the server
I	:	size of each index entry
B	:	broadcast data rate

Table 1: Marginal Gain Parameters

Let us first present how the marginal gain for an item i is computed when promoting it to vapor or demoting it to liquid. The parameters are shown in table 1. Note, that the marginal gain in both cases is computed similarly except for the sign of the involved quantities. Therefore, to avoid duplication in the presentation of the algorithm we introduce the variable A which takes the value -1 if the item i is vapor and considered for demotion to liquid and $+1$ if it is liquid and considered for promotion to vapor. The computations are based on the model described in section 2.1. The only difference is that now we also take into account the overhead of broadcasting the index. For simplicity, we assume that a copy of the index is broadcast with every item.

We compute the expected overall marginal gain dT as a weighted average of the marginal gains dT_{push} and dT_{pull}. If we define $d\Lambda_V = A\,\lambda_i$ we have:

$$dT = \frac{\Lambda_V\, dT_{push} + \Lambda_L\, dT_{pull}}{\Lambda_V + \Lambda_L}$$

[4]Similarly to the vaporization temperature of water which depends on the atmospheric pressure.

where

$$dT_{push} = A\,\frac{S_i + (2N_V + A)\,I}{2\,B}$$

and

$$dT_{pull} = T_{pull}\,\frac{d\Lambda_V}{\mu - \Lambda_L + d\Lambda_V}$$

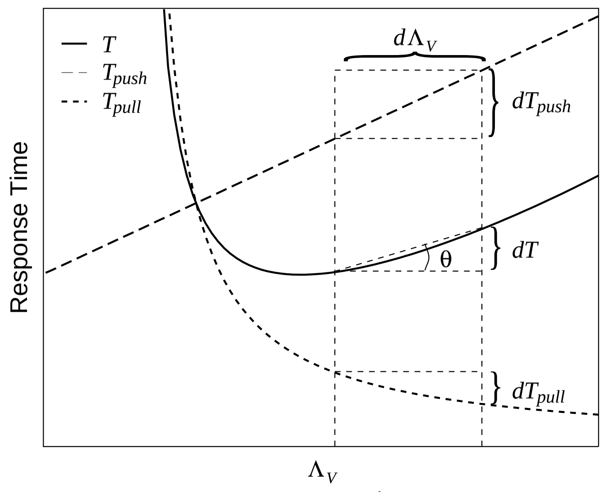

Figure 3: Marginal gains

Figure 3 depicts these computations graphically. Ideally, by using the marginal gains the systems should try to reach and operate at the minimum point of the curve T. However, it turns out that in practice this is not the best thing to do. This is explained by the fact that to the left of this minimum point the response time grows very fast. As a result, under a dynamic workload it is very probable that even a small change can have a very bad effect on the system. Therefore, operating at or too close to the minimum can make the system very unstable. Later in the paper, this is indeed verified by our experiments. So, instead, we have to force the system to operate in a suboptimal area to the right of the minimum, safely avoiding instability. We achieve this by establishing some small (but not zero) threshold θ_0 for the angle $\theta = tan^{-1}\frac{dT}{d\Lambda_V}$.

In figure 4 we outline the actual algorithm that uses these marginal gains to update the contents of the vapor queue $\mathcal{V}$. Intuitively, the algorithm performs three simple steps: First, it demotes to liquid all vapor data whose temperature is smaller than that of the hottest liquid item. Then, using the respective marginal gains, it continues demoting vapor items in increasing order of temperatures while $\theta > \theta_0$. Last, it takes the opposite direction and as long as $\theta < \theta_0$ it promotes liquid data to vapor in decreasing order of temperature. Note that if at least one vapor

```
$l$ = hottest liquid item
$v$ = coldest vapor item
while ($\lambda_v < \lambda_l$)
        demote $v$
        $v$ = coldest vapor item
end
while ($-dT/\lambda_v > tan\theta_0$)
        demote $v$
        $v$ = coldest vapor item
end
$l$ = hottest liquid item
while ($dT/\lambda_l < tan\theta_0$)
        promote $l$
        $l$ = hottest liquid item
end
```

Figure 4: Adaptive algorithm based on marginal gains

item is demoted in the second step, then no liquid item will be promoted in the third step. Also, it is possible that vapor items that get demoted in the first step will be re-promoted in the third.

Figure 5 illustrates an example of how the algorithm works. We assume that initially items A, B, C, D, E, F, and G are vapor, items H and I are liquid, and that $\lambda_A \leq \lambda_B \leq \lambda_C \leq \lambda_D \leq \lambda_E \leq \lambda_F \leq \lambda_I \leq \lambda_G \leq \lambda_H$. In this case, the algorithm firsts demotes A, B, C, D, E, F, and G since their temperature is smaller than that of the liquid H. Then it detects that there is no further gain by demoting more items so it skips the second step. At the third step it promotes three items, H, G, and I (G was temporarily demoted in the first step).

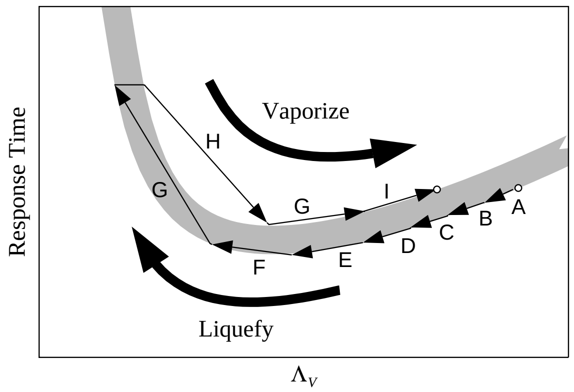

Figure 5: Execution of the adaptive algorithm

3.4 Probing for Snappy Reflexes

A potential weakness in what has been described so far could be the artificial cooling down of vapor data. It was introduced for the sole purpose of giving the server a chance to re-evaluate the temperature of vapor data regularly, and generally it is not expected to reflect the real-life evolution of data demand. In that sense, it may very well result in a situation where a very hot item is demoted to liquid because its artificially cooled down temperature has declined. If that happens, it is very possible that the server will be swamped with requests for that item. Even though, the adaptive algorithm will eventually correct this by re-promoting the item, the reaction time lag may be big enough to cause serious degradation. This section introduces "probing" for achieving snappy reflexes to rapid workload changes.

This is explained better in the left part of figure 6 where we present the time line of events after a decision to demote a vapor hot item. At t_0, the server decides to demote a vapor item and sends a message to notify the clients about its decision. This notification is done with the broadcast of the the next index message which arrives at t_1. From that point on, all the requests for that item are directed to the server. If the item is still hot, the server decides again to make it vapor at t_2 and notifies the clients again with a message that arrives at t_3. The shaded area in the figure represents the total request load that such wrong decisions may generate. Considering data transmission and server inertia delays (i.e. the time it takes the server to re-promote the item), the interval between t_1 and t_3 could be substantial. Therefore, the cumulative effect of consecutive improper data demotion can be enough to make the system practically unusable.

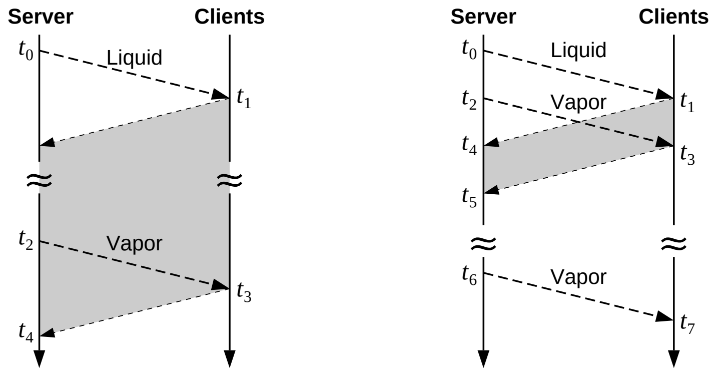

Figure 6: Temperature re-evaluation without and with probing

The adaptive algorithm that we propose remedies this potential error by a *double clutch probing* approach, which is illustrated on the right hand side of figure 6. Soon after the decision to convert an item from vapor to liquid at t_0, and way before it is actually heated up by misses, the item is temporarily re-promoted at time t_2. This creates a controllably small time window (from t_1 to t_3) that limits the expected number of client requests for the demoted item, but still can provide valuable information to the server. In effect, we give the server the opportunity to "probe" for the actual temperature of the data just before committing to its decision. After the forced promotion of the item at t_2, the server waits for any requests generated during the period t_1–t_3 in order to re-evaluate the item's actual temperature. Due to network inertia, we delay this re-evaluation until some time after t_5. At that time (t_6), depending on its probed temperature, the item is either demoted or reinstated to the broadcast queue with accurate temperature.

The power of double-clutch probing is that it can reduce the size of the shaded area, which corresponds to misses, in a controllable manner. However, the selection of the probing interval t_0–t_2 is very critical. If it is too short, hardly any requests will be generated to help the server in the re-evaluation. If it is too long, the benefits of the probing are lost. In addition, the probing time should be dynamically adjusted to the intensity of the workload. For these reasons, we found that a very good selection can be based on the average request rate of vapor data. More specifically, we set the probing time to be proportional to the inverse of the average temperature of vapor data. In this way, we can explicitly control the expected number of requests each probing window allows, and, consequently, the total probing overhead.

3.5 Monitoring Overhead

The implementation of our hybrid scheme requires some considerable bookkeeping which may by itself impose a heavy computational load on the server. The server needs to monitor the temperature of liquid items, keep them sorted so that each time the next candidate for promotion can be identified instantly, as well as detect those that have not received any attention for a period long enough to freeze.

To keep this overhead down to a minimum, we chose to organize bookkeeping around the idea of *slotted time*, which considers time as divided into slots $S_0, S_1, \ldots, S_n, \ldots$, each taking time t_s. During each slot, we record the total number of requests for every item that gets requested. We then compute the request rate of each item using a moving average over time slots exponentially weighted by a factor α. Formally, the request rate of item i at the end of slot n is $\lambda_{i,n} = \alpha\, r_{i,n}/t_s + (1-\alpha)\, \lambda_{i,n-1}$ where $r_{i,n}$ is the number of requests for item i made during S_n.

The computational benefits of this approach are two-fold. First, note that for the items that were not requested during the last slot we have $r_{i,n} = 0$, and therefore $\lambda_{i,n} = (1 - \alpha)\,\lambda_{i,n-1}$. In practice, the server does not even need to update these values, since for an item last requested during S_{n-k} it holds that $\lambda_{i,n} = (1-\alpha)^k\,\lambda_{i,n-k}$. This way we avoid many computations which are performed only when (if ever) needed. The second benefit is that the relative order of temperatures for items not requested during the last slot does not change. This is exploited to significantly reduce the overhead of keeping a list of liquid items in decreasing order of temperatures. Only the items requested in the last slot need to be sorted according to their new computed temperatures and then quickly merged with the rest.

Last, with the time slots it is straightforward to identify when liquid items become frigid. Assume that an item freezes if it is not requested for l slots $t_l = l\,t_s$. Then, at the end of slot S_n, the items that were last requested during S_{n-l} turn into frigid. As a result, each time we need to keep information only about the last l time slots.

4 Experiments and Results

4.1 Simulation model and parameters

In order to establish the potential of hybrid data delivery, investigate the involved trade-offs and explore the possible alternatives, we have built a simulation model of the proposed system. We model wide area information systems where a very large number of clients (in the order of thousands) access an information source. For the simulations, we have assumed that the provided information is a collection of N self-identifying data items each of equal size. For our purposes, every client generates requests for data which are satisfied either by the broadcast program or the server upon explicit request. Under this assumption, we have modeled this large client population as a single module that generates the total workload, a stream of independent requests for data items. The exact number of clients is not specified but instead it is implicitly suggested by the aggregate request rate.

In all our experiments we used two different workload distributions: *HotColdUniform* and *Gaussian* (see figure 7. These distributions proved to be very useful for our investigation. The first is only used as an ideal case where there is a clearly defined hot-spot in the database. The second is far more realistic, but at the same time it allows the explicit customization of the

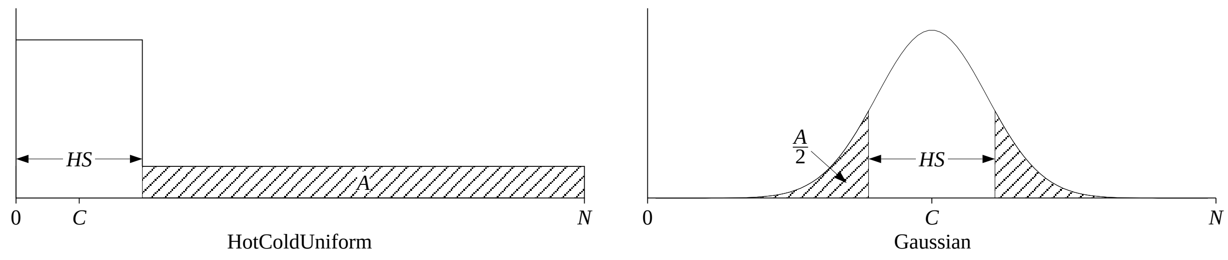

Figure 7: Workload distributions

hot-spot size and the tail area through parameters. Actually both distributions are defined by the same parameters, namely RR the aggregate request rate, A the aggregate request rate for cold data (i.e. the tails of the distribution), HS the width of the hot-spot in terms of data items and C the center of the hot-spot. In order to create the effect of dynamic workloads, these parameters can be variable in the experiments. More specifically, we allow RR, HS, and C to vary periodically within certain ranges. For example, RR changes from RR_{min} to RR_{max} within some selected period Π_{RR}. Obviously, by setting $RR_{min} = RR_{max}$ we achieve some constant RR. HS is controlled similarly. For C we define some sort of speed at which it moves in the data set. Note that the fact that the hot items are continuous in the database does not affect our results. Any permutation of data will yield the same results.

For the server we have used a simple data server model, enhanced with a transmitter capable of broadcasting and the functionality required to implement our adaptive algorithm. As usually, we assume a fixed amount of memory cache and explicit requests are serviced in FIFO order unless blocked for I/O. Although the server is modeled through several parameters (e.g. cache size, I/O characteristics, e.t.c.), the presentation and interpretation of our results is based only on one parameter, the server throughput, which corresponds to the maximum rate at which requests can be serviced. Depending on the experiment setup, this is determined by (a combination of) the bandwidth of the available network, the size of the data, and the processing power of the server.

The last part we model is the network on which the system operates. Since we would like to capture the behavior of asymmetric hybrid environments we need to specify the characteristics of three communication paths: (1) the broadcast channel, (2) the downlink from the server to the clients, and (3) the uplink from the clients to the server. For simplicity, we assume that all clients use similar but independent paths for establishing point-to-point connections with the server (e.g. regular or cellular phone lines). The downlink on the other hand is a shared resource that is used for all server replies. The broadcast channel, although physically can coexist with the downlink, logically it is considered a separate channel with a fixed specially allocated bandwidth. For each of the communication paths, we define a data transmission rate and a propagation delay. We must also note that in our study so far we have ignored communication errors. Dealing with the interesting problems that arise in the presence of errors is among the goals of our future work.

In the following subsections we present the most interesting results of our experiments. Although many more experiments were performed, in this paper we present only the most important characteristics about the system's performance.

4.2 Static Workloads

For the first set of experiments we chose to use static workloads, even though static ones do not demonstrate how quickly the system adapts to these workloads. However, they can provide a solid base for comparison since for those we can determine the optimal behavior of a hybrid delivery system. Actually, the graphs in this section include two baselines for comparison. The first, marked "Theory" represents the theoretically optimal policy, based on the the model of section 2.1. For the second, marked "PerfectServer", we have obtained an exhaustive algorithm that finds the optimal broadcast program and used a stripped version of our server which does not adapt but instead, it broadcasts periodically the optimal program. Essentially, under static workloads the "PerfectServer" line is the ultimate performance goal of our system.

For the experiments presented in this section we have assumed the parameters of the table 2. These correspond to a hybrid architecture like the DirecPC [Hug] of Hughes Network Systems. It can be easily computed that the broadcast and the downlink rates are roughly 30 items per second. Assuming that enough computing power at the server, this is also the maximum service throughput.

Broadcast Rate	:	12 Mbps
Downlink Rate	:	12 Mbps
Uplink Rate	:	28.8 Kbps
Data Item Size	:	50 KB
Database Size	:	10000 Items

Table 2: Simulation Parameters

The basic metric for the system is the average response time for the clients. We intend to show that under the workload and system assumptions discussed in section 2.2 our system can exhibit very high scalability. In other words, it can accommodate very heavy workloads with no significant degradation in the clients response time. In the experiments, we vary the workload from very light (below the server's maximum throughput) to very heavy (100 times the server's maximum throughput), with a fixed hot-spot size of 100 items (1% of the database).

First, we present the results we obtained under the ideal workload distribution *HotColdUniform.* In figure 8 we show the average response time for this case as a function of the total request rate. For completeness, we include the performance of a pure data-pull system (no broadcast). As expected, such a system cannot accommodate any workload higher than its capacity (approximately 30 requests per second in this case). On the other hand it is clearly shown that the hybrid delivery approach as implemented in our system can easily scale to workloads 100 times heavier or even more (note that the horizontal axis is in logarithmic scale). No matter how heavy the workload gets the response time remains constant, roughly equal to half the time it takes to broadcast the hot-spot. Obviously, this depends only on the size of the hot-spot. Moreover, under this ideal separation of hot and cold data our approach performs optimally under heavy workloads, matching both the theoretically minimum response time and that of the perfect server. For lighter workloads, it performs slightly worse since it chooses to broadcast few items even though it is not necessary. This is a result of the introduction of the threshold θ_0 (see section 3.3) which makes our algorithm slightly biased towards broadcasting.

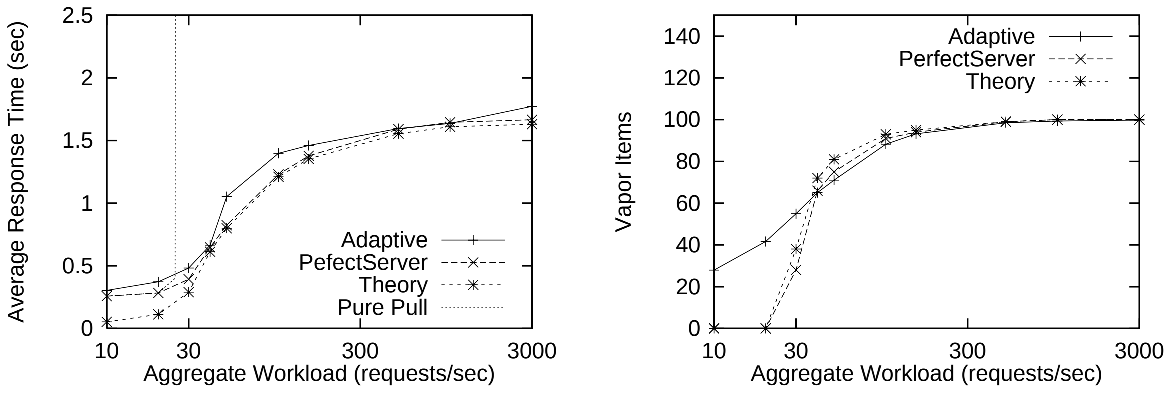

Figure 8: HotColdUniform Workload Distribution

In the second graph of figure 8 we show the average number of items that our system had in vapor state. Comparing that to the theoretically optimal number of vapor items, we see that our system selects the same number of data to broadcast and indeed these are the best ones. Although, this result is based on the ideal workload distribution, it is expected to hold in every type of workload where the hot items can be clearly distinguished.

However, in many realistic situations this is not true. So, in order to test our system under "continuous" distributions, where the boundaries of the hot-spot are not clearly defined, we performed another set of experiments using Gaussian workload distribution. All the workload and system parameters are the same as in the previous experiments. In figure 9 we show the response time of the system, together with the average number of vapor items. Again, the most impressive result is the scalability that the system achieves. For workloads one hundred times more than the system's capacity the response time remains very small. However, this time, there is a small yet distinguishable difference between the performance our system achieves and the theoretically optimal. From the right graph we can infer that this difference is attributed to the fact that our algorithm does more broadcasting than the optimal according to the theoretical model and the perfect server. This again is a result of the small tendency of our algorithm to favor broadcasting, which now detects, outside the optimal hot-spot, items hot enough to be included in the broadcast. This was not the case in the previous distribution. A related interesting thing to note is that the selected hot-spot appears to be improving (leaning towards the optimal) as the aggregate rate increases. This is attributed to the fact that the tail of the workload in every case remains below the server's capacity and, as the total workload increases, the distribution becomes more skewed. As a result, the boundaries of the hot-spot are more accurately identified by our algorithm.

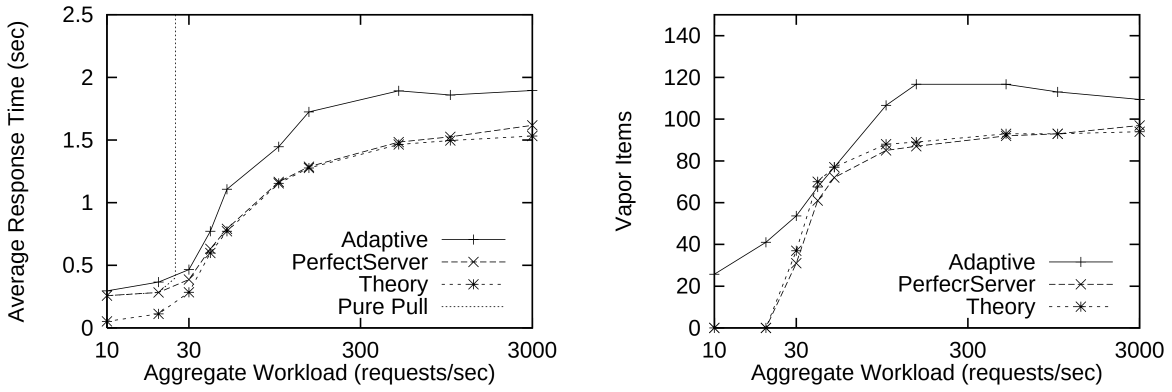

Figure 9: Gaussian Workload Distribution

4.3 Sensitivity Analysis

In this section we present results that demonstrate the effects of the major parameters that control the performance of the system. In the discussion of section 3.3 we talked about the threshold θ_0 that we need to establish in order to save the system from instability. We ran a series of experiments to show the actual effects of that threshold. In figure 10 we show a representative example from these experiments. Note that the results are presented as a function of $tan\theta_0$ which corresponds to a threshold for the more intuitive quantity $dT/d\Lambda_V$. These graphs verify that small thresholds bring the system closer to the optimal but lead to instability. As can be observed from the left graph of figure 10, the curve breaks sharply for values less than 0.08. Thus a small perturbation on the threshold has explosive negative consequences on the performance. Instead, in all the experiments we ran we noticed that a threshold of 0.1 yields suboptimal but acceptable performance, by making the server broadcast slightly more. Greater values for this threshold increase the broadcast even more with no extra benefit. The right graph of the same figure shows the extra broadcasting caused by the threshold.

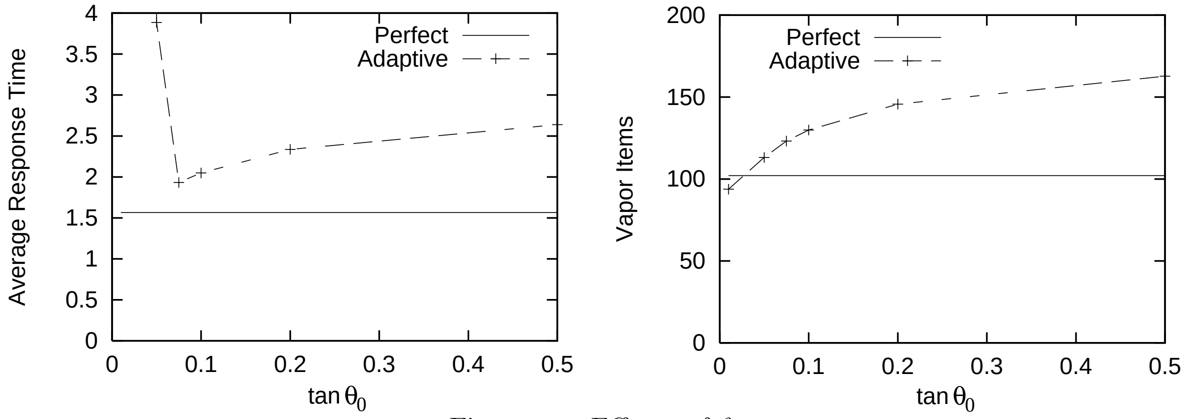

Figure 10: Effects of θ_0

An equally important part of our system is the ability to probe the workload. In turns out that direct demotion of very hot vapor items to a liquid state can be detrimental to the system performance. As was mentioned earlier, the selection of the probing time is very important. If it is either too small or too big, it is essentially the same as no probing at all. This is clearly demonstrated in figure 11. According to section 3.4 the probing time should be proportional to the inverse of the average vapor request rate. For our experiments, we define the probe time to be equal to that quantity multiplied by a "Probe Time Factor". This way, we directly control the number of expected requests per probe, i.e. for a factor of 4 we get an average of 4 requests per probe. In all the experiments, we noticed that we are getting the best results for values between 3 and 6, which means that on average 3 to 6 requests are enough for estimating data temperatures and not too many to cause problems.

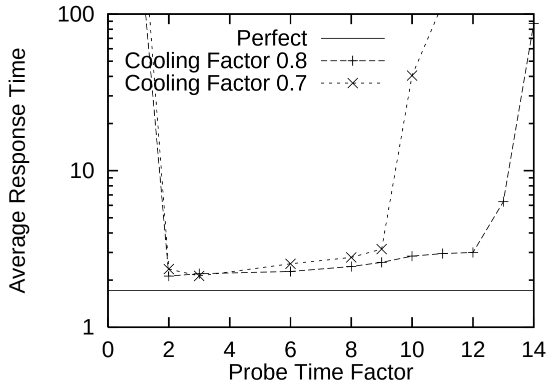

Figure 11: Effects of probing

In figure 11 we show the effect of probing time for two different values of the cooling factor. The reason for this experiment is that the cooling factor implicitly controls the frequency of probing and thus affects its performance. The trade-off is that more frequent probing generates higher overhead in terms of broadcast misses while infrequent probing hinders the adaptiveness of the system. The larger the factor and the more frequent the probing the greater the overhead, which explains why for a cooling factor of 0.7 the range of acceptable values for the probe factor is smaller.

Overall, we can say that the power of our approach is that, with a proper combination of these two parameters, one can explicitly control fairly accurately the adaptiveness of the system, the effectiveness of the probing as well as the incurred overhead.

4.4 Dynamic Workloads

For the last set of experiments we test the adaptiveness of our system under dynamic workloads. We simulate dynamic workloads by defining a "speed" for the hot–spot of the database. For example, a hot–spot speed of 1000 items/per hour means that within one hour the hot–spot shifts gradually 1000 items. For the results presented here we assume that the other parameters of the workload do not change. In figure 12 we present the performance results of the system for speeds varying from 0 (static workload) to 5000 items per hour. As expected, the cooling factor is crucial to the adaptation speed of the system so we give results for three different values.

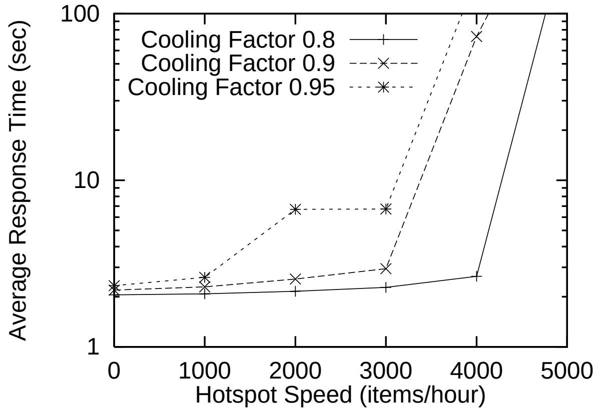

Figure 12: Adaptation under dynamic workload

From the performance graph of figure 12 we see that the system, when properly tuned, can adapt even to very fast changing workloads. It can detect shifts in the clients interest for data, and change the content of the broadcast almost without any noticeable delays. The power of the approach lies in the ability to probe the workload as often as necessary with small and controllable overhead. Therefore, any errors in the estimation of the workload can be quickly corrected with minimum penalty.

A final note about our approach is that it is possible to employ a self–tuning strategy for the system. In other words, the system can monitor the workload behavior and the result of its previous actions to teach itself as to how it should be operating more efficiently. As an example, if after a series of probing the result is always the same, it may be good idea to increase the cooling factor and sample less frequently. Or on the other hand, if most of the times probing suggests that vapor items are cooling down very fast, it may advisable to decrease the cooling factor so even faster adaptation can be achieved.

5 Related Work

The use of asymmetric connectivity in hybrid network architectures has been pioneered by the Center for Satellite and Hybrid Communication Networks (CSHCN) at the University of Maryland, since 1990, as the most feasible technologically and economically efficient means for the development of a National and Global Information Infrastructure (NII/GII). The CSHCN and Hughes Network Systems (HNS) co-developed hybrid networking products based on satellites [FAS$^+$95] which demonstrated this very concept.

Data broadcasting has been a research area for more than a decade. Early work was done in the context of teletext systems [AW85] and database machines [HGLW87]. More recently, with the introduction of mobile computing and extensive wireless communication services, it has gained much more research [IB94, FZ96] and commercial attention (e.g. [Air]). In terms of research the focus has been in optimized broadcast schedules [AAFZ95, ST97] based on precompiled knowledge about access patterns, optimized techniques for data retrieval from a broadcast channel [AAFZ95, AFZ96], and power efficiency considerations for mobile environments [IVB94b, IVB94a].

The issue of hybrid data delivery has received less attention. The Boston Community Information System (BCIS) developed by Gifford [GBBL85] was a pioneering project in terms of combining *simplex* (broadcast) and *duplex* (interactive) systems to build a scalable and flexible information system. The systems used an FM channel to periodically broadcast updates (news articles) to locally maintained user databases. User queries were answered by the local database whenever possible. If the requested information was not available locally, the system would query the server through a regular modem connection. The system was implemented and tested by about 200 users for a period of two years. According to [Gif90], the major conclusions of the experiment were that users value both components of the hybrid architecture, and that this approach is a very economic way to building large scale information systems.

The work of Wong and Dykeman [WD88] proposes and analyzes a hybrid teletext-videotext architecture. Their results emphasize the need for an adaptive scheme that can select the appropriate delivery mode for the data. In [IV94], Imielinski and Viswanathan propose an adaptive algorithm for wireless information systems that, similarly to our system, combines broadcast and unicast data delivery. However, their assumptions and goals are quite different. Based on fairly static access probabilities, they are proposing a way to assign data and bandwidth to the two deliver modes that guarantees a maximum expected response time. Last, Acharya, et al. are exploring a sort of hybrid approach in the context of the Broadcast Disks project. In [AFZ97] they augment their push-only architecture by allowing clients to explicitly request data for expedite delivery through the same broadcast channel.

6 Conclusions

In this paper, we described an adaptive algorithm for hybrid networks that takes advantage of broadcast channels for massive data dissemination and unicast channels for data delivery not satisfied by the former. The algorithm continuously adapts the broadcast content to match the hot-spot of the workload. We showed, that the hot-spot can be accurately obtained by monitoring the "broadcast misses" and, thus, no knowledge on the actual usage of the broadcast data is necessary. This is one of the major distinctions between the work presented here and all other broadcast schemes which are totally dependent on accurate but not available statistics on full workload access patterns.

We have shown that the adaptive scheme performs effectively even under very dynamic and rapidly changing workloads. Probing was introduced to increase the sensitivity of the system and make its reflexes snappy. We showed that the speed of adapting to the workload's characteristics can be controlled with probing which adapts the size of sampling for broadcast data. Finally, we characterized the workloads that would best work in such a hybrid data delivery environment by showing that the overall system's throughput depends only on the size of the hot-spot and not on the volume of the workload. This has far reaching implications for very large scale and high volume wide area information systems.

There is a lot of future work to be done. We are exploiting the use of multiple broadcast and unicast channels to deal with various sizes of data items,and/or control data, the use of redundant broadcast programs, overlapping data broadcast, etc. We are also exploiting forecasting for prefetching and "on-time" data delivery.

Acknowledgments

We would like thank Björn T. Jónsson, Flip Korn, Yannis Kotidis, Alexandros Labrinidis, and Christos Seretis for their invaluable help. Björn's semi-automatic results compiler proved to be a great time saver.

References

[AAFZ95] Swarup Acharya, Rafael Alonso, Michael J. Franklin, and Stanley B. Zdonik. Broadcast Disks: Data Management for Asymmetric Communications Environment. In

Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 199–210, San Jose, California, May 1995.

[AFZ96] Swarup Acharya, Michael J. Franklin, and Stanley B. Zdonik. Prefetching from Broadcast Disks. In Stanley Y. W. Su, editor, *Proceedings of the 12th International Conference on Data Engineering*, pages 276–285, New Orleans, Louisiana, February 1996.

[AFZ97] Swarup Acharya, Michael J. Franklin, and Stanley B. Zdonik. Balancing Push and Pull for Data Broadcast. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, May 1997. to appear.

[Air] AirMedia. AirMedia Live. http://www.airmedia.com.

[AW85] M. Ammar and J. Wong. The Design of Teletext Broadcast Cycles. *Perfomance Evaluation*, 5(4):235–242, December 1985.

[FAS+95] A.D. Falk, Vivek Arora, Narin Suphasindhu, Douglas Dilon, and John S. Baras. Hybrid Internet Access. In *Conference on NASA Centers for Commercial Development of Space*, number 325 in AIP Conference Proceedings, pages 69–74, New York, 1995.

[FZ96] Michael J. Franklin and Stanley B. Zdonik. Dissemination-Based Information Systems. *IEEE Bulletin of the Technical Committee on Data Engineering*, 19(3):20–30, September 1996.

[GBBL85] David K. Gifford, Robert W. Baldwin, Stephen T. Berlin, and John M. Lucassen. An Architecture for Large Scale Information Systems. In *Proceedings of the Tenth ACM Symposium on Operating System Principles*, pages 161–170, Orcas Island, Washington, December 1985.

[Gif90] David K. Gifford. Polychannel Systems for Mass Digital Communications. *Communications of the ACM*, 33(2):141–151, February 1990.

[HGLW87] Gary E. Herman, Gita Gopal, K. C. Lee, and Abel Weinrib. The Datacycle Architecture for Very High Throughput Database Systems. In Umeshwar Dayal and Irving L. Traiger, editors, *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, pages 97–103, San Francisco, California, May 1987.

[Hug] Hughes Network Systems. DirecPC. http://www.direcpc.com.

[IB94] Tomasz Imielinski and B.R. Badrinath. Wireless Mobile Computing : Challenges in Data Management. *Communications of the ACM*, 37(10):18–28, October 1994.

[IV94] Tomasz Imielinski and S. Vishwanathan. Adaptive Wireless Information Systems. In *Proceedings of SIGDBS (Special Interest Group in DataBase Systems) Conference*, Tokyo, Japan, October 1994.

[IVB94a] Tomasz Imielinski, S. Viswanathan, and B. R. Badrinath. Energy Efficient Indexing on Air. In Richard T. Snodgrass and Marianne Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 25–36, Minneapolis, Minnesota, May 1994.

[IVB94b] Tomasz Imielinski, S. Viswanathan, and B. R. Badrinath. Power Efficient Filtering of Data on Air. In Matthias Jarke, Janis A. Bubenko Jr., and Keith G. Jeffery, editors, *4th International Conference on Extending Database Technology*, pages 245–258, Cambridge, United Kingdom, March 1994.

[SRB96] Konstantinos Stathatos, Nick Roussopoulos, and John S. Baras. Adaptive Data Broadcasting Using Air–Cache. In Tzi cker Chiueh, editor, *First International Workshop on Satellite-based Information Services*, pages 30–37, Rye, New York, November 1996.

[ST97] C.-J. Su and L. Tassiulas. Broadcast Scheduling for Information Distribution. In *Proceedings of IEEE INFOCOM'97*, Kobe, Japan, April 1997. to appear.

[WD88] J.W. Wong and H.D. Dykeman. Architecture and Performance of Large Scale Information Delivery Networks. In *12th International Teletraffic Congress*, Torino, Italy, 1988.