Very Large Data Bases

Proceedings of the Twenty-third International Conference on Very Large Data Bases

EDITORS: Matthias Jarke
Michael Carey
Klaus R. Dittrich
Fred Lochovsky
Pericles Loucopoulos
Manfred A. Jeusfeld

Athens, Greece
26-29 August, 1997

Adaptive Data Broadcast in Hybrid Networks*

Konstantinos Stathatos
kostas@cs.umd.edu

Nick Roussopoulos
nick@cs.umd.edu

John S. Baras
baras@isr.umd.edu

Institute for Systems Research, University of Maryland

Abstract

With the immense popularity of the Web, the world is witnessing an unprecedented demand for data services. At the same time, the Internet is evolving towards an information super-highway that incorporates a wide mixture of existing and emerging communication technologies, including wireless, mobile, and hybrid networking. Taking advantage of these new technologies, we are proposing a hybrid scheme which effectively combines broadcast for massive data dissemination and unicast for individual data delivery. In this paper, we describe a technique that uses the broadcast medium for storage of frequently requested data, and an algorithm that continuously adapts the broadcast content to match the hot-spot of the database. We show that the hot-spot can be accurately obtained by monitoring the "broadcast misses" observed through direct requests. This is a departure from other broadcast-based systems which rely on efficient scheduling based on precompiled user profiles. We also show that the proposed scheme performs effectively even under very dynamic and rapidly changing workloads. Extensive simulation results demonstrate both the scalability and versatility of the technique.

1 Introduction

The world is witnessing an unprecedented demand for data services. The immense popularity of the Web is generating 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 have already been observed in the course of special events such as the 1996 Olympics, the last elections in the US, etc. when several million requests were made during peak periods.

Traditional unicast (point-to-point) 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 it would be underutilized and wasted during non-peak periods. On the other hand, the Internet is evolving towards an information super-highway that incorporates a wide mixture of communication technologies, including wireless, mobile, and hybrid networking[KB96, BG96, Kha97]. In this environment, new types of information services are surfacing and practical solutions to the anticipated explosion of user demands are being proposed[FZ96]. Among these, broadcast-based services have the potential of meeting such workloads, as they can efficiently disseminate information in a connection-less mode to any number of receivers, with no significant performance degradation in terms of access latency. But a major concern for the success of such systems is broadcasting the right set of data, i.e. data for which there is indeed vigorous demand.

In [SRB96], we introduced a hybrid system that effectively combines broadcast for massive data dissemination (*broadcast data push*) and unicast for upon-request data delivery (*unicast data pull*). This system is built around the notion of *air-caching*, i.e. the use of the available broadcast capacity for temporary storage of frequently requested data. The key issue is the identification of the database hot-spots to be air-cached, so that only a small load of "broadcast misses" is left to be serviced in the usual connection-oriented way. There are, however, at least two major obstacles: First, data needs can be neither characterized nor predicted a-priori because of the dynamic nature of the demand. For example, emergency or weather related situations may cause abrupt shifts in demand. Therefore, techniques based on precompiled broadcast schedules are not applicable in this case. Second, users receiving information from a broadcast channel are passive, in the sense that they do not communicate with the server to acknowledge the usefulness of the broadcast. Therefore, the server lacks a lot of invaluable information about actual data needs.

*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

Proceedings of the 23rd VLDB Conference
Athens, Greece, 1997

In this paper, we propose a technique that continuously adjusts the broadcast content to match the hot-spot of the database. We show that the hot-spot can be accurately obtained by monitoring the "broadcast misses" observed through direct requests. This is a departure from all other schemes which rely on complete—but in most cases unavailable—knowledge about 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 a unicast-only server by at least two orders of magnitude, under the assumption that the server's capacity is sufficient for servicing the cold set of data. If that holds, the performance of the hybrid system proposed herein is independent of the total volume of the workload and, thus, the system exhibits significant scalability, even for rapidly changing access patterns.

2 Hybrid Data Delivery

In this section, we develop a simplified analytical model for hybrid data delivery which will provide some intuition behind our work and illustrate the involved trade-offs. Based on this model, we discuss how broadcast and unicast can work synergistically to yield high data service rates.

2.1 The Hybrid Model

In a hybrid scheme, we can exploit the characteristics of each of the data delivery modes and integrate them in a way that better matches the clients' demands. The objective is to deliver the needed data with minimum delay to very large numbers of clients. Striving for that goal, we should be looking for solutions that range between:

Pure broadcast (push)[1]: The server repetitively (periodically) broadcasts all the data items, while clients are passive listeners who make no requests. The repetition allows the broadcast medium to be perceived as a special memory space. Its major advantage is that it can be accessed concurrently by any number of clients without any performance degradation. This makes broadcasting an attractive solution for large scale data dissemination. However, its limitation is that it can be accessed only sequentially, as clients need to wait for the data of interest to appear on the channel. A direct consequence is that access latency depends on the volume of the data being broadcast, which has to be fairly small.

Pure unicast (pull): This is a standard client-server architecture where all requests are explicitly made to the server. Such a scheme cannot scale beyond the capacity of the server and the network. The average data access time depends on the aggregate workload as well as the network load, but not on the size of the database.

[1] Throughout this paper the terms data push and broadcast are used interchangeably; so are the terms data pull and unicast

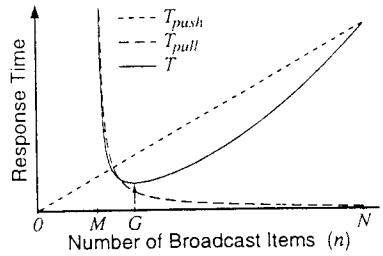

Figure 1: Hybrid Delivery Trade-off

Consider a database containing N data items of equal size S. Assume that the demand for each item i forms a Poisson process of rate λ_i with the items numbered such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N$. A server, modeled as an M/M/1 system, services requests for these items with mean service time $1/\mu$. In addition, this server can broadcast data over a channel at a rate B. Also assume that, for some reason, the server decides to broadcast the n first items and offer the rest on-demand. If we define $\Lambda_k = \sum_{i=1}^{k} \lambda_i$, then the expected response time for requests serviced by the server is $T_{pull} = \frac{1}{\mu - (\Lambda_N - \Lambda_n)}$, while for those satisfied by the broadcast it is $T_{push} = \frac{n\,S}{2\,B}$, half the time required to broadcast all n items. The expected response time T of the hybrid system is the weighted average of T_{pull} and T_{push}.

Figure 1 plots a representative example of T, T_{pull} and T_{push} as a function of n, i.e. the number of broadcast items. We have assumed that the total workload is greater than μ, which is a safe assumption for large scale systems with huge client populations. Henceforth, we refer to μ as the system's *pull capacity*. The first thing to note in this figure is that the performance of the pull service T_{pull} 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 to the volume of the broadcast data. The slope of that line is determined by the size of the data S and the available bandwidth B. Hence, too much broadcasting is not desirable either. Obviously, for 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 about Workloads

The discussion of the previous section suggests that it is possible to balance data delivery modes in order to obtain optimal response time. However, this optimal solution depends on the shape and size of the imposed workload. In what follows, we explore hybrid delivery from a practical perspective and give a qualitative answer to how a combi-

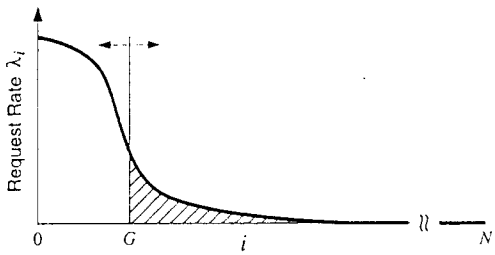

Figure 2: Skewed Data Access Pattern

nation of broadcasting and unicasting can be advantageous.

Intuitively, data broadcasting is helpful when its content is useful to multiple receivers. The benefit is twofold: first, with each broadcast message the server saves several unicast messages that otherwise would have to be sent 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 hardly any receivers do not yield any benefit,[1] but instead harm overall performance by occupying valuable bandwidth. This implies that broadcasting is effective when there is significant commonality of reference among the client population. Ideally, we would like 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 becomes 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 pulled. The area to the left of G (the head of the distribution) represents the volume of requests satisfied by the broadcast. The shaded area to the right of G (the tail of the distribution) represents the volume of the explicit requests directed to the server. According to the model presented in the previous section, the response time depends on the area of the tail and the width of the head (i.e. the number of broadcast items). The height of the head reflects the savings of broadcasting. Generally, the selection of G should satisfy two constraints:

(1) The tail should be maintained below the pull capacity.
(2) The head should be wide enough to accommodate the hot-spot but should not include rarely requested data.

While the first constraint is intuitive, the second deserves some clarification, as it is critical to the practicality of a hybrid solution. Consider a case where the tail is a very long area of very small, but not zero, height. That represents a large number of items that each gets requested very infrequently. If this area is larger than the pull capacity, 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 the quality of the broadcast content would substantially deteriorate by including lots of rarely requested items, yielding unacceptably high response time, which nonetheless would be optimal according to our model. Consequently, under such workloads, slightly increased pull capacity is a more favorable solution than inordinate broadcasting.

Bearing this in mind, we consider cases where the optimal solution does not require broadcasting rarely requested data. It is assumed that the pull capacity is at least such that it can handle the aggregate load imposed by requests for such data. Under this assumption, we propose an adaptive hybrid scheme that, 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.

[1] Assuming there is another way to satisfy those very few receivers

3 Adaptive Hybrid Delivery

In this section we elaborate on the proposed adaptive hybrid delivery scheme. Our approach 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. Much like typical cache memories, this *air-cache* is used to increase throughput in terms of requests serviced per time unit, and should be adaptive to changing workloads. The challenge in making it adaptive lies in the fact that the server cannot have any information about "air-cache hits" simply because they are not acknowledged by the clients. Therefore, traditional cache management techniques based on cache hits, such as LRU, MRU, etc., are not applicable. Instead, the algorithm presented here relies on "air-cache misses", indicated by explicit requests for data not broadcast, that provide the server with tangible statistics on the actual demand. 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 hits on the broadcast, the more satisfied the clients.

3.1 Vapor, Liquid and Frigid Data

In our work, for each item in the database we define a *temperature* that corresponds to its request rate λ_i. In addition, each item can be in one of three possible states (for a more intuitive presentation, we borrow terminology from their analogy to the physical states of water):

Vapor: Items deemed as heavily requested which are therefore broadcast, i.e. put in the air-cache.

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

Frigid: Items that have not been requested for a while and their temperature λ_i has practically dropped to 0.

In the proposed adaptive scheme, the server dynamically determines the state of the database items, relying on air-cache misses. These can be considered as the "sparks" that regulate the temperature and state of the data. Specifically:

- Vapor data are retrieved from the air-cache, and the server does not get any feedback about their actual temperature. As they are not heated by requests, they gradually cool down and eventually turn into liquid. The duration of the cooling process depends on the temperature that initially turned them into vapor.
- 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, again depending on the intensity of the requests. Obviously, as long as they get no requests they remain frigid.

The hardest part of this process is distinguishing vapor from liquid data, and this is the focus of this paper. The distinction between liquid and frigid data items is the same to that achieved by a buffer manager of a database system using a frequency-based replacement policy[RD90, OOW93]. Likewise, the server should maintain liquid items in main memory anticipating new requests in the near future, and can retrieve frigid items from secondary memory only when necessary. In practice, the distinction of frigid data plays an important role in terms of overhead, specially in the case where frigid data make up the largest part of the database. With a default 0 temperature, the server is off-loaded from tracking their demand statistics, and can also safely ignore them when looking for candidate vapor items.

3.2 Repetitive Data Broadcasting

In order to create the effect of caching on the air, we employ a repetitive broadcast scheme. Contrary to typical periodic broadcast schemes that assume a fixed schedule, the size and content of our broadcast is continuously updated to better match the workload. The heart of our approach is a queue $\mathcal{V}$ which stores all vapor data. The server picks the next item to broadcast from the head of $\mathcal{V}$. After an item gets broadcast, it is removed from the head and gets appended back to tail of $\mathcal{V}$. At the same time, its temperature is multiplied by a predetermined $CoolingFactor \in (0, 1)$ to reflect the cooling process of vapor data.

The contents of $\mathcal{V}$ are modified once every cycle, the end of which is identified by a vapor item specially assigned as a placeholder. Once this placeholder is broadcast, the server re-evaluates the state of data and updates the queue accordingly. In this adaptation process, described in detail in the next section, it pinpoints vapor items that should be demoted to liquid, and liquid items that need to be promoted to vapor. Vapor items selected for demotion are marked, so that after their next broadcast they will be removed from the queue. New vapor items are placed on the tail of queue. Finally, the (new) item on the tail of $\mathcal{V}$ is assigned as the next placeholder. The result is a repetitive broadcast scheme with evolving size and content.

An integral part of the hybrid delivery scheme is the indexing of the air-cache. Since clients are expected to select between the two data delivery paths, the server needs to make them aware of items forthcoming in the broadcast channel. Here, we have adopted a simple technique that uses the signature of $\mathcal{V}$ (i.e. the list of data identifiers in the queue) as an index that is broadcast 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 broadcast frequency of the index can be adjusted to trade overhead for the maximum time clients are willing to wait before making the decision. Note that, depending on the size and the number of vapor items, it is possible that this simple indexing scheme will yield considerable overhead. For such cases, we plan to investigate more elaborate indexing schemes, such as bit-vectors or the schemes proposed in [IVB94a] and [IVB94b].

3.3 Adaptation Based on Marginal Gains

In this section, we present the algorithm that adapts the contents of the broadcast. As we already mentioned, in the adaptation process, the server needs to make two kinds of decisions: (a) which of the vapor data have cooled down enough to be demoted to liquid, and (b) which of the liquid data have become hot enough to be promoted to vapor. A straightforward approach of establishing absolute temperature thresholds cannot be applied because the state of an item depends also on the aggregate workload, i.e. the relative temperature of the other items. To account for that, we have developed an algorithm that makes these decisions based on the expected marginal gain of each possible action.

Let us first present how the expected marginal gain is computed when considering an item i for promotion to vapor state or demotion to liquid. Note that in both cases it is computed similarly except for the sign of the involved quantities. Therefore, to avoid duplication in the presentation, we use 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. The additional variables used here are the aggregate request rate for liquid data Λ_L, the aggregate request rate for vapor data Λ_V, the number of vapor items N_V, and the size of each index entry I. The expected overall marginal gain dT is given by the weighted average of the marginal gains dT_{push} and dT_{pull} where, if we define $d\Lambda_V = A\,\lambda_i$, we have:

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

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

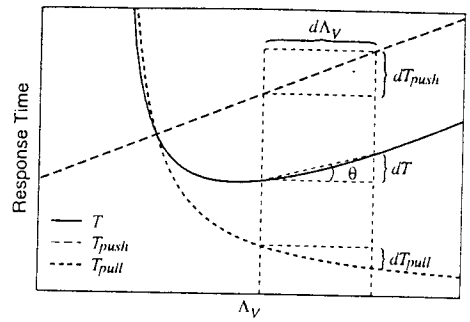

Figure 3: Marginal gains

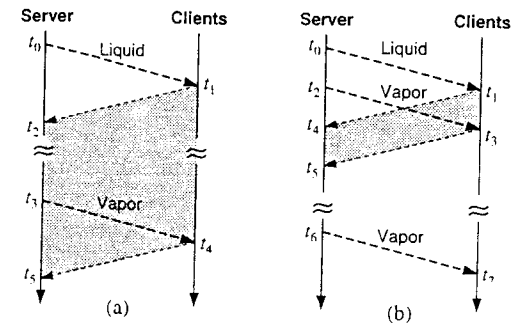

Figure 5: Demotion (a) without, and (b) with probing

Figure 3 depicts these computations graphically. Ideally, the system 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. This was indeed verified by our experiments[SRB97]. 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 $\theta = tan^{-1}\frac{dT}{d\Lambda_V}$.

The actual algorithm that updates the contents of the vapor queue $\mathcal{V}$ consists of three simple steps: First, it demotes to liquid all vapor data with temperature lower than 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 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. If data items are sorted by their temperatures,[2] the complexity of this algorithm is in the order of the number of items that change state.

Figure 4 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,

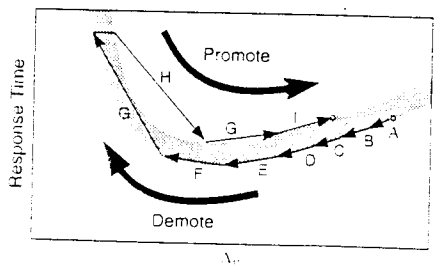

Figure 4: Example execution of the adaptive algorithm

the algorithm firsts demotes A, B, C, D, E, F, and G since their temperature is lower 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 demoted in the first step).

3.4 Temperature Probing

A potential weakness in what has been described so far is the artificial cooling 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, Thus, it is not expected to reflect the actual evolution of data demand, and may very well result in a situation where a very hot item is demoted to liquid. Should that happen, the server will be swamped with hundreds or thousands of 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 performance degradation.

This is better explained in Figure 5a where we present the time line of events after a decision to demote a hot vapor item at time t_0. This decision is reflected in the next broadcast of the index which reaches the clients 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 re-promote it to vapor at t_3, and includes it at the next index broadcast, received by the clients at t_4. But, considering data transmission and server inertia delays (i.e. the time to re-promote the item), the interval between t_1 and t_4 could be substantial. The shaded area in the figure represents the total request load that this wrong decision may generate. The cumulative penalty of consecutive improper demotions can be heavy enough to make the system practically unusable.

This section introduces *temperature probing* as a way of preventing any disastrous effects by premature demotions of vapor data. The algorithm that we propose remedies potential errors by a "double clutch" approach, which is illustrated in Figure 5b. Soon after the decision to convert an item from vapor to liquid at t_0, and before it is actually heated up by misses, the item is re-promoted at time t_2. This creates a controllably small time window (from t_1

[2] In [SRB97] we discuss how to maintain the order with low overhead

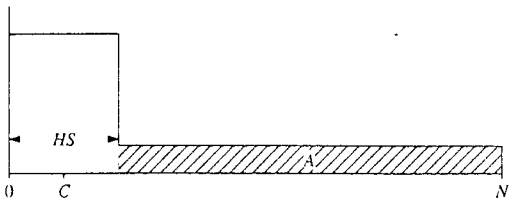

Figure 6: HotColdUniform distribution

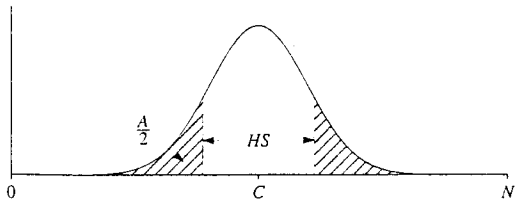

Figure 7: Gaussian distribution

to t_3) that limits the expected number of client requests for the demoted item, but still can provide the server with concrete information about the actual demand. In effect, through a small number of misses, we give the server the opportunity to probe for the actual temperature of the data, before committing to its decision. After the re-promotion of the item at t_2, the server waits for 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 at least until t_5. Finally, depending on the result of the probing, the item is either demoted or reinstated to the broadcast queue with corrected temperature at t_6.

A critical factor for this double-clutch approach is the probing interval $[t_0, t_2]$. If it is too short, hardly any requests will be generated to help the server in the re-evaluation. If it is too long, it essentially defeats its purpose. Therefore, it should be selected very carefully, and should preferably 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 $ProbingFactor \times \frac{N_V}{\Lambda_V}$, where $\frac{N_V}{\Lambda_V}$ is inverse of the average temperature of vapor data. Essentially, with this demand-adjusted probing window, the *ProbingFactor* determines the expected number misses generated per probe, and allows the system to explicitly control the total probing overhead.

4 Experiments and Results

4.1 Simulation Model

In order to establish the potential of hybrid data delivery and investigate the involved trade-offs, we have built a simulation model of the proposed system. For our experiments, we assumed that the provided information is a collection of self-identifying data items of equal size (e.g. disk pages). Clients generate requests for data which are satisfied either by the broadcast or the server upon explicit request. Under this assumption, we have modeled all the 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. For the data access pattern we used two different distributions: *HotColdUniform* and *Gaussian* (Figures 6 and 7). The first one is only used as an ideal case where there is a clearly defined hot-spot in the database. The second is more realistic, but at the same time it allows explicit customization through the same four parameters: the aggregate request rate *RR*, the aggregate request rate for cold data *A*, the width of the hot-spot in terms of data items *HS*, and the center of the hot-spot *C*. In order to create the effect of dynamic workloads, the value of these parameters can vary in the course of an experiment. For example, by changing the value of *C* we can simulate workloads with moving hot-spots.

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. Although it is modeled in detail through several parameters (e.g. cache size, I/O characteristics, etc.), the presentation and interpretation of our results is based only on one parameter, the system's pull capacity μ, 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 processing power of the server, the available bandwidth, and the size of the data. For the network, since we want to capture 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. The downlink on the other hand is a shared resource that is used for all server replies. The broadcast channel is considered a separate channel with a fixed specially allocated bandwidth. We must also note that in our study so far we have ignored communication errors.

4.2 Static Workloads

For the first set of experiments we used static workloads, even though they cannot demonstrate the system's adaptiveness. The reason is that they can provide a solid base for comparison, since for those we can easily 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, based on the the model of Section 2.1. For the second, marked "PerfectServer", we used a stripped version of our server which does not adapt, but instead, broadcasts periodically the optimal set of data, obtained through exhaustive search. For static workloads, the line "PerfectServer" is the ultimate performance goal of our system.

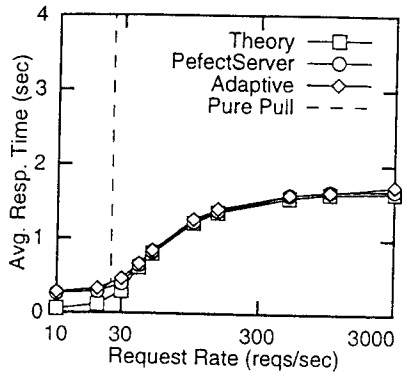

Figure 8: HotColdUniform

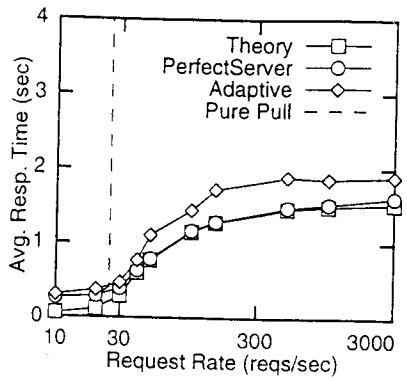

Figure 9: Gauss, Fixed hot-spot

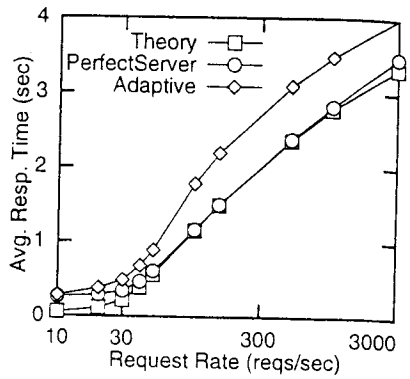

Figure 10: Gauss, Expanding hot-spot

For the experiments presented in this section we have assumed that both broadcast and downlink rates are 12Mbps, while the uplink is 28.8Kbps. These could correspond to a hybrid architecture like the DirecPC[Hug]. The database consists of 10000 items, each 50KB in size. For this data size, the broadcast and the downlink capacities are roughly 30 items per second. Assuming enough computing power at the server, this is also the system's pull capacity μ. For the workload, we vary its volume from light ($RR < \mu$) to very heavy ($RR = 100 \times \mu$) to show the behavior of the system in different scales. We intend to demonstrate that, under the assumptions discussed in Section 2.2, our approach performs close to the optimal, and exhibits very high scalability. A significant performance property of this system is that response time depends only on the size of the hot-spot, and not on the intensity of the workload.

First, we present the results we obtained under the ideal *HotColdUniform* workload distribution. In Figure 8, we show the average response time as a function of the request rate *RR*. The size of the hot-spot *HS* remained constant (100 items) for all values of *RR* in order to highlight the above mentioned property. For contrast, we include the performance of the pure pull system which, as expected, cannot accommodate workloads higher than its capacity ($\approx$ 30 requests/sec). On the other hand, it is evident that the hybrid delivery approach can scale to workloads at least 100 times heavier (note that the horizontal axis is in logarithmic scale). Even for large values of *RR*, response time remains practically constant. Moreover, under this ideal separation of hot and cold data, our approach performs optimally, matching both the theoretically minimum response time and that of the perfect server. As *RR* grows, most of the requests become air-cache hits, and therefore, the average response time is dominated by the performance of the hits. Obviously, this depends only on the size of hot-spot size, and is roughly equal to half the time it takes to broadcast it. The load incurred by air-cache misses is maintained below the pull capacity, consistently yielding sub-second responses.

Next, in order to test our system under more realistic workloads, where the boundaries of the hot-spot are not clearly defined, we performed a set of experiments using the Gaussian distribution. All the workload and system parameters are the same as in the previous case. However, now we present results obtained under both fixed and expanding hot-spot sizes. For increasing values of *RR*, a fixed hot-spot was achieved by increasing accordingly the skewness of the distribution (i.e. decreasing its standard deviation); for the expanding hot-spot we used a constant standard deviation. Figure 9 shows the performance under a fixed hot-spot, where again we see that even for large workloads the response time remains very small. However, this time there is a small, yet distinguishable, discrepancy between our system and the optimal. The reason is that our system selected to broadcast, on the average, a few more items over what both the theoretical model and the "PerfectServer" suggest as optimal. This is an artifact of the threshold θ_0 (Sec. 3.3) which urges the adaptive algorithm to slightly favor broadcasting. Contrary to the previous case, the algorithm now detects, outside the optimal vapor set, items hot enough to be considered for promotion. Figure 10 presents the results for the same experiment, but with expanding hot-spot. From the two baselines, we can see that the optimal vapor set—and consequently the optimal response time—is indeed growing with *RR*. But, even in this case our system scales very well, in the sense that it manages to follow the optimal performance very closely.

4.3 Tuning Parameters

In Section 3, we introduced three important tuning parameters, namely θ_0, *CoolingFactor*, and *ProbingFactor*. While the first is used just to keep the system at a safe distance away from instability, the other two are essentially the knobs that control its adaptiveness and overhead. Here, we concentrate on the effects of the latter two parameters. For θ_0, we have established from previous experiments that a good a selection is such that $\frac{dT}{dN_V} = tan\theta_0 \geq 0.1$ [SRB97].

Temperature probing was introduced to prevent the detrimental consequences of early demotions of vapor items, but the probing window needs to carefully selected; if it is either too small or too big, it is essentially the same as no probing

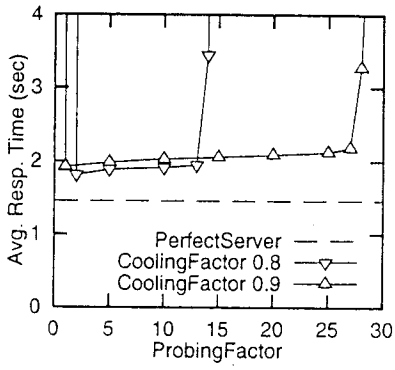

Figure 11: Effects of probing

at all. In Section 3.4, we defined the probing window to be dynamically adjusted by the *ProbingFactor* and the average request rate for vapor items. This way, we directly control the number of expected misses per probe, i.e. for a *ProbingFactor*=4 we get an average of 4 requests per probe. The *CoolingFactor* (Sec. 3.1) is also very related to probing, and must be carefully selected as well. A small value causes the temperature of vapor data to drop quickly, yielding frequent probing and high overhead in terms of probed misses. But, on the positive side, a small value also allows the system to adapt faster to changes in the demand. Large values have the opposite effect; they cause less probing but hinder the adaptiveness of the system.

Figure 11 shows how the *ProbingFactor* affects the system's performance, for two different values of the *CoolingFactor*. For this experiment, and the rest of the experiments presented hereafter, we used the Gaussian access pattern with *RR*=500 requests/sec and *HS*=100 items. The first thing we note is that, without probing (*ProbingFactor*=0), the system cannot recover from the incorrect demotions, and the response time grows arbitrarily large. But even a very small number of probed misses (≥ 2) are sufficient to correct the temperatures of vapor data, thus allowing the system to operate close to the optimal. As the *ProbingFactor* increases further, so does the volume of the probed misses. The rate at which this happens depends on the frequency of the probing (i.e. the *CoolingFactor*) and the number of items being probed (i.e. the number of vapor items). Beyond some point, the overhead of probed misses becomes too big for the server to handle, leading again to very slow responses. In other words, with a very large *ProbingFactor*, probing causes the problem that it was supposed to solve in the first place. Naturally, this happens earlier when probing is more frequent (*CoolingFactor*=0.8).

4.4 Dynamic Workloads

For the last set of experiments, we used dynamic workloads in order to evaluate the adaptiveness of our system in cases when the focus of the clients' demand changes. Such a change was modeled as an elimination of a hot-spot and a generation of a new one in another (randomly selected) part of the database. This process was not instant, but instead it was taking a transient period of *TP* minutes to complete. Every new hot-spot persisted for *Duration* minutes. For easier interpretation of the results, all the hot-spots were similar, and the total workload remained constant.

In Figure 12 we present the obtained results as a function of *Duration*. The workload in these graphs is more dynamic on the left side, since with smaller *Duration* changes occur more often. We used two different values of *TP* for comparing fast (white marks, *TP*=2min) and more gradual (black marks, *TP*=5min) changes. Also, we give results for two values of the *CoolingFactor* (*CF*=0.9 and *CF*=0.8) which determines the adaptation speed of the system. For better comprehension of the results, we graph the total average response time (Fig. 12a), the average response time for pulled data (Fig. 12b), and the average number of vapor items (Fig. 12c). For all these experiments, we used *ProbingFactor*=5 and $tan\theta_0$=0.1.

The most significant observation is that the system adapts very well to changing access patterns (Fig. 12a). Even on the left side where changes occur very frequently, the response time remains small. It most cases, performance lies within 1 sec of that achieved under the static workload (Fig. 9). This means that the server is very effective in detecting shifts in the clients demand, and thus can react promptly. As expected, the system adapts and performs better with a smaller *CoolingFactor*. But, an unexpected result shown in Figure 12a is that the system appears to perform better under more abrupt changes (*TP*=2min). However, this will be justified in the following where we discuss how the system is affected by dynamic workloads.

Changing hot-spots have a performance impact on both the pull (Fig. 12b) and the push (Fig. 12c) part of the system. First, an item that suddenly becomes hot can generate a large number of requests before the server is able to react and append it to the air-cache. The cumulative effect of these requests may cause significant build-up in the server's input queue, and therefore increase the average response time for pulled data. This build-up is worse when the changes are faster and more frequent. Indeed, in Figure 12b we see that the average pull response time increases when the changes occur more often (left side) and when new hot-spots are heating up faster (white marks). Second, in transient periods the server actually perceives two hot-spots, the old and the new. Thus, in order to meet the demand during those periods, it has to expand the vapor set to include them both. This explains why in Figure 12c the average number of vapor items increases as the *Duration* decreases. With decreasing *Duration*, the transient periods make up more and more of the total time. As a result, the server appears to be broadcasting, on the average, more data. Note that for *Duration*=*TP*=5min the workload is continuously in transient state and the server almost always detects two hot-spots. As a result, the size of vapor set is close to double that of the static case. We also observe that this

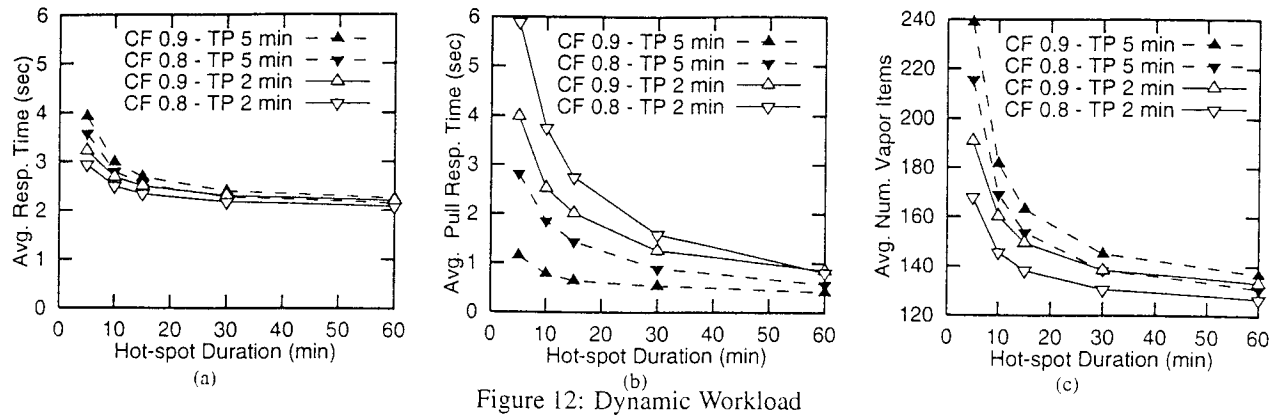

Figure 12: Dynamic Workload

phenomenon is worse with longer transient periods (black marks) as the server spends more time broadcasting both hot-spots. Since, in these experiments, the average response time is dominated (≈90%) by broadcast accesses, this also explains why the system appears to perform better under more abrupt changes (*TP*=2min).

Finally, here we can also notice the effects of the *CoolingFactor* to the adaptiveness of the system. On one hand, the smaller value (*CF*=0.8) harms the pull response time since it causes more frequent probing and, thus, more misses (Fig. 12b). But, on the other hand, it limits unnecessary broadcasting and reduces the "double hot-spot" phenomenon since it allows the server to detect faster loss of interest for vapor data (Fig. 12c). Consequently, the *CoolingFactor* should be selected as small as it causes tolerable probing overhead. Note that the probing overhead can be estimated (and controlled) by the *CoolingFactor*, the *ProbingFactor*, and the number of vapor items. Also, it is even possible to employ a self-tuning strategy for the system. In other words, the system can monitor the workload behavior and the results of its previous actions to learn how it should be operating more efficiently. As an example, if after a series of probings the outcome is always the same, it may be good idea to increase the *CoolingFactor*and sample less frequently. Overall, one of the strongest features of this approach is that, with a proper combination of two parameters, we can explicitly control fairly accurately the adaptiveness of the system, the effectiveness of the probing, and the incurred overhead.

5. Related Work

The idea of broadcasting or pushing data from some information source to a large number of receivers is being studied for more than a decade. Early work was done in the context of teletext and videotex systems[AW85, Won88], community information services[GBBL85, Gif90], as well as specialized database machines[HGLW87, BGH+92]. More recently, with the proliferation of wireless communication and mobile computing, it has gained much more research[IB94, FZ96] and commercial attention (e.g. [Poi, Air]). In terms of research, the focus has been mostly in optimized broadcast schedules[AAFZ95, ST97], optimized techniques for data retrieval from a broadcast channel[AAFZ95, AFZ96], and power efficiency considerations for mobile environments[IVB94b, IVB94a].

Hybrid data delivery was first employed in the Boston Community Information System[Gif90] which combined broadcast and interactive communication to provide up-to-the-minute information to an entire metropolitan area. A prototype was built and was field-tested for a period of two years by about 200 users. The major conclusions of this experiment were that users valued both components of the hybrid architecture, and that this approach is indeed a very economic way to building large scale information systems. A hybrid teletext-videotex architecture was proposed in [WD88]. Their approach involved only broadcast delivery, but for both periodically pushed and upon-request pulled data with some ad hoc partition of the data into two groups. The same combination of delivery modes was considered in [AFZ97]. In particular, they augmented the push-only architecture of broadcast disks[AAFZ95], by allowing clients to explicitly request data for expeditious delivery through the same broadcast channel. In that work, they explore the efficacy of a back-channel in a broadcast-only environment and discuss the involved trade-offs.

Even closer to out work, are the adaptive techniques proposed in [IV94] and [DCK+97]. [IV94] proposes an algorithm that, based on fairly static access probabilities, assigns data and bandwidth to broadcast and on-demand delivery modes in a way that limits the maximum expected response time below a predefined threshold. Last in [DCK+97], they consider mobility of users between cells of a cellular network, and propose two variations of an adaptive algorithm that statistically selects data to be broadcast based on user profiles and registrations in each cell.

6 Conclusions

In this paper, we described an adaptive technique for hybrid data delivery that takes advantage of broadcast channels for massive data dissemination and unicast channels for data

demand not satisfied by the former. We first discussed how broadcast and unicast can work synergistically to yield high data service rates, and then presented an algorithm that, based on marginal gains and broadcast probing, continuously adapts the broadcast content to match the hot-spot of the database. We showed that the hot-spot can be accurately obtained by monitoring the "broadcast misses" and therefore no other implicit 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 dependent on accurate, comprehensive, but not readily available statistics on workload access patterns.

Our simulation experiments have demonstrated both the scalability and versatility of the proposed technique. Under the assumption that the server's capacity is sufficient for servicing the demand for cold data, it performs very close to the optimal. An important result of is that the performance of this hybrid system depends only on the size of the hot-spot, and not on the volume of the workload. We have also shown that this adaptive scheme performs very well even under dynamic, rapidly changing workloads. The adaptation speed and the incurred overhead can be explicitly tuned as desired.

We believe that these results have far reaching implications, as they suggest an effective way of deploying large scale wide area information systems. Therefore, there is a lot of interesting work to be done in the near future. We are currently exploring many different issues including client querying, dealing with data of various sizes, multi-frequency broadcasting, efficient indexing schemes, overlapping data broadcast, forecasting for prefetching, and "on-time" data delivery.

Acknowledgments

We would like thank Michael Franklin, 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] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast Disks: Data Management for Asymmetric Communications Environment. In *Proc. ACM SIGMOD Conf.*, pages 199–210, May 1995.

[AFZ96] S. Acharya, M. Franklin, and S. Zdonik. Prefetching from Broadcast Disks. In *Proc. of the 12th Intl. Conf. on Data Engineering*, pages 276–285, February 1996.

[AFZ97] S. Acharya, M. Franklin, and S. Zdonik. Balancing Push and Pull for Data Broadcast. In *Proc. ACM SIGMOD Conf.*, pages 183–194, May 1997.

[Air] AirMedia. AirMedia Live. *www.airmedia.com.*

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

[BG96] G. Bell and J. Gemmell. On-ramp Prospects for the Information Superhighway Dream. *CACM*, 39(7), July 1996.

[BGH+92] T. Bowen, G. Gopal, G. Herman, T. Hickey, K. C. Lee, W. Mansfield, J. Raitz, and A. Weinrib. The Datacycle Architecture. *CACM*, 35(12):71-81, Dec 1992.

[DCK+97] A. Datta, A. Celik, J. Kim, D. VanderMeer, and V. Kumar. Adaptive Broadcast Protocols to Support Efficient and Energy Conserving Retrieval from Databases in Mobile Computing Environments. In *Proc. of the 13th International Conference on Data Engineering*, pages 124–134, April 1997.

[FZ96] M. Franklin and S. Zdonik. Dissemination-Based Information Systems. *IEEE Data Engineering Bulletin*, 19(3):20–30, September 1996.

[GBBL85] D. Gifford, R. Baldwin, S. Berlin, and J. Lucassen. An Architecture for Large Scale Information Systems. In *Proc. of the 10th ACM Symposium on Operating System Principles*, pages 161–170, December 1985.

[Gif90] D. Gifford. Polychannel Systems for Mass Digital Communications. *CACM*, 33(2):141-151, Feb. 1990.

[HGLW87] G. Herman, G. Gopal, K. Lee, and A. Weinrib. The Datacycle Architecture for Very High Throughput Database Systems. In *Proc. ACM SIGMOD Conf.*, May 1987.

[Hug] Hughes Network Systems. DirecPC. *www.direcpc.com.*

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

[IV94] T. Imielinski and S. Vishwanathan. Adaptive Wireless Information Systems. In *Proc. of SIGDBS Conf.*, Tokyo, Japan, October 1994.

[IVB94a] T. Imielinski, S. Viswanathan, and B.R. Badrinath. Energy Efficient Indexing on Air. In *Proc. ACM SIGMOD Conf.*, pages 25–36, May 1994.

[IVB94b] T. Imielinski, S. Viswanathan, and B.R. Badrinath. Power Efficient Filtering of Data on Air. In *Proc. EDBT Conf.*, pages 245–258, March 1994.

[KB96] R. Katz and E. Brewer. The Case for Wireless Overlay Networks. In *SPIE Multimedia and Networking Conference*, San Jose, CA, January 1996.

[Kha97] B. Khasnabish. Broadband To The Home (BTTH): Architectures, Access Methods and the Appetite for it. *IEEE Network*, 11(1):58–69, Jan./Feb. 1997.

[OOW93] E. O'Neil, P. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm For Database Disk Buffering. In *Proc. ACM SIGMOD Conf.*, pages 297–306, May 1993.

[Poi] PointCast, Inc. PointCast Network. *www.pointcast.com.*

[RD90] J. Robinson and M. Devarakonda. Data Cache Management Using Frequency-Based Replacement. In *ACM SIGMETRICS Conf.*, pages 134–142, May 1990.

[SRB96] K. Stathatos, N. Roussopoulos, and J.S. Baras. Adaptive Data Broadcasting Using Air–Cache. In *1st Intl. Workshop on Satellite-based Information Services*, November 1996.

[SRB97] K. Stathatos, N. Roussopoulos, and J.S. Baras. Adaptive Data Broadcast in Hybrid Networks. Technical Report 97-40, Institute for Systems Research, Univ. of Maryland, Apr. 1997.

[ST97] C.J. Su and L. Tassiulas. Broadcast Scheduling for Information Distribution. *IEEE INFOCOM'97*, Apr. 1997.

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

[Won88] J. Wong. Broadcast Delivery. *Proceedings of the IEEE*, 76(12):1566–1577, December 1988.