To be presented in the First International Workshop on Satellite-based Information Services (WOSBIS), Rye, New York, November 13, 1996

Adaptive Data Broadcasting Using Air–Cache*

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

Abstract

In the Data AirWaves Project at University of Maryland, we are integrating Direct Broadcast Satellite (DBS) systems with terrestrial networks to provide a hybrid and effective communication substrate lying between data resources and remote/mobile user applications. Smooth integration of these two media balances the need for rapid data dissemination to very large numbers of clients and on–demand interactive data services. This paper describes the air–cache, a method for effective data broadcasting and an algorithm which rapidly adapts the content of the cache based on the "misses" which result in explicit (on–demand) data requests. Simulation results show that the hypothesis of adapting based only on the misses performs quite reasonably and has very little deviation from a system that has complete information – both hits and misses.

*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 Dep. of Computer Science

‡Also with the Dep. of Electrical Engineering

1 Introduction

Wireless connectivity is becoming increasingly important. Applications utilizing satellite wireless networks are emerging in the areas of multimedia communications and mobile computing [10]. Both of these areas are characterized by their increasing requirement for data to be "here and now". Direct Broadcast Satellite Systems (DBS) provide a very effective communication substrate lying between the data resources and the remote/mobile user applications.

DBS systems are particularly attractive for networks with huge client population because they result in potentially unlimited cumulative bandwidth and reduce or eliminate individual client requests, data transmissions and the associated overhead [6]. However, passive DBS systems are limited because clients have no means of communicating neither their data needs which can be dynamically changing, nor how useful the content of the broadcast is. Neither hits nor misses on the content are reported.

On the other hand, Interactive Data Service (IDS) connectivity in which clients connect through some (wireless) terrestrial networks for on–demand *data pull*, can build a "fully–informed" model of the clients' needs.

Monitoring the data pull allows the server to improve performance by either caching the hot–spots or prefetching correlated access patterns.

In this paper we combine DBS with terrestrial (wireless) IDS access to strike a balance between the need for rapid data dissemination and the need for specificity/filtering of requests. We use DBS to broadcast the hot set and, thus, achieve the highest cumulative bandwidth of broadcast, while we monitor the misses from IDS requests. The rates of the misses are used to promote the data objects to the broadcast channel (make them hot). We show that the dynamic behavior of the data need can accurately be estimated by monitoring the misses through the IDS.

1.1 Existing Approaches

A small number of research projects have recently addressed some similar issues. Generally, when it comes to data broadcasting the crucial questions are:

What to broadcast? For environments without up–links (i.e. clients cannot transmit) the whole database has to be broadcasted [2]. The main problems of this approach are: (1) the database may be too big for the available bandwidth, (2) bandwidth may be wasted for broadcasting data never used, (3) it is static. If, however, up–links are available the system can choose to broadcast only the most frequently requested data [7]. But, techniques proposed so far rely mostly on some apriori knowledge about access probabilities which are assumed to be fairly static.

When to broadcast? Two general techniques have been proposed. The first is probabilistic data selection, i.e. select the object to broadcast next using a predetermined randomized function based on data access probabilities [11, 7]. The main drawback is that access time may grow arbitrarily large (starvation problem). The second is periodic or cyclic broadcasting of (a selected set of) data [2]. It guarantees a maximum access time equal to the broadcast period, and it may be optimal in terms of minimizing the mean response time [11].

How to broadcast? Different structures for the broadcast program have been proposed, each optimizing different performance criteria. The simplest is flat broadcasting under which a set of self–identifying data objects are broadcasted sequentially. Clients have to listen to the broadcast channel until the object of interest arrives. Broadcast disks [1] improve average data access time, by grouping data into popularity groups and broadcasting each such group with different frequency. Proper client data caching and prefetching techniques can compensate for infrequently broadcasted data and mismatching access probabilities [3]. Last, special attribute indexing techniques that interleave data and (primary and secondary) index structures (e.g. trees, hash tables) have been proposed as a way of reducing client tuning time and energy consumption [8, 9].

2 Effective Data Broadcasting: Air–Cache

Let us assume that we broadcast data with period T over a channel of bandwidth B. This broadcast can be considered to form a memory space of size $B \times T$ with some special characteristics:

- It can be accessed by any number of clients concurrently, i.e. there is no access contention.
- It can be accessed only sequentially. A direct consequence is that the average

access time depends on the size of the memory which in turn is determined by the period T.

- The server cannot have any information about the effectiveness of this memory space, i.e. which – if any – clients actually use it.

The most important question is to find out when data broadcasting is effective. In striving for the best overall performance, we should be looking for solutions in the range between:

Broadcast everything (pure data push): This scheme includes the case when for one reason or another clients are or choose to be completely passive making no explicit requests. Such a scheme can accommodate an arbitrary large number of clients (unlimited scalability), although, the average data access time may grow with the size of the database and, thus be unacceptably high.

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

We define the data broadcast scheme to be effective if it provides data to the clients faster and/or at a smaller cost compared to an explicit request to the server, and at the same time reduces contention for data access at the server improving the system's scalability margin. This can be achieved if the server can maintain a good balance between "data push" and "data pull". In other words, the goal should be to broadcast the right amount of the hottest data that would satisfy the bulk of the clients' concurrent requests and leave the rest of the requests to be serviced explicitly.

At the same time, we would like to guarantee the best overall throughput performance for any given workload even when it is very dynamic and changes substantially. This is exactly what is needed simply because:

- typically clients unpredictably connect to and/or disconnect from the system
- mobile clients arbitrarily join or leave coverage areas.
- data request patterns are not static over time. For example, in the morning users usually need information about traffic delays and the weather, while in the evening they may want to know about movie showing times or table availability in local restaurants!
- unscheduled events may generate bursty requests for relevant information (e.g. emergencies, news, sport results).

These are the very same performance objectives achieved with data caching. Therefore, we can treat the broadcast capacity as a global cache memory between the server and the clients. This "air–cache" should be adaptive to the system workload in order to allow clients to get the data they need faster and/or at a smaller cost than directly from the server. The challenge in making adaptive data broadcasting effective lies on the facts that the server cannot have a clear picture about the actual usage of the broadcasted data simply because satisfied clients do not acknowledge (at least not in real time) the usefulness of the received data. "Air–cache misses" indicated by explicit requests for not broadcasted data objects, provide the server the statistics on their demand frequency. Thus, the more misses the better server statistics. But, on the other hand, the more passive the clients are, the better are satisfied with the broadcast.

The air–cache can be adapted in three ways:

Size: Given a channel bandwidth, the size of the air–cache is determined by the broadcasting period, which in turn determines the average access time.

Contents: Assuming that the entire database cannot be broadcasted within a single period, the server has to decide what data should be broadcasted.

Program: The program determines the order and the structure of the broadcasted data. Many issues that affect client performance can be taken into account. For example, should some hot data be replicated in the cache (i.e. broadcasted more than once in one period as in broadcast disks) in order to reduce their access time? How should data be structured and/or indexed so that the client tuning time is minimal?

3 Adaptive Air–Caching

Our goal is to implement an *adaptive air–cache* utilizing a repetitive data broadcasting scheme. Note that we use the term repetitive instead of periodic since, although data are broadcasted repeatedly, there is no clear fixed broadcast period. The air–cache is adapted to the current workload in order to reduce both average data access time and the number of explicit data requests.

3.1 Vapor, Liquid and Frigid Data

A key idea of our approach is to define for each object in the database a *temperature* which corresponds to its current request rate λ. Based on their temperature, objects can be in one of three states:

Vapor (Steamy) Hot: Very hot objects (intensively requested) which are air–cached (i.e. broadcasted).

Liquid Warm: Objects with lower request rate, not large enough to justify broadcasting but still sufficient to require fast access from the server. For that reason, liquid data are kept in the server's main memory buffers.

Frigid (Icy) Cold: Objects that get requested very infrequently and, therefore, their temperature λ is practically 0 (degrees centigrade). In many applications, frigid data comprise the bigger part of the database and are maintained in secondary or even tertiary memory.

For the proposed adaptive scheme, the server needs to dynamically determine the state of the database objects. The only information available to it which provides some insight about data need are the "air–cache misses", i.e. the explicit requests made to the server for data not in the air–cache. These can be considered as the "sparks" that regulate the temperature and the state of the data.

In general, the following rules control the data states:

- Frigid data that start being requested may turn into liquid or even vapor depending on the intensity of their sparks. Obviously, as long as they get no sparks they remain frigid.
- Liquid data that get requested either turn into vapor or remain liquid, again depending on the intensity of the sparks. Liquid data that stop being requested eventually freeze.
- We assume that clients always prefer accessing data from the broadcast channel

whenever possible (i.e. the data they request are broadcasted)[1]. Therefore, there are no sparks for vapor data which are gradually cooling down until they turn into liquid again (and provided on demand thereafter). The time it takes for them to cool down depends on the temperature that caused them to vaporize in the first place.

3.2 Adaptive Repetitive Data Broadcasting

In order to maximize the effectiveness of broadcasting, the server needs to continuously update the size and contents of the air–cache to best match the ever changing system workload. In other words, it should keep selecting the set of objects to vaporize. Obviously, the selection should be based on the decreasing order request rates, i.e. the hottest objects first. The number of vapor objects (i.e. the size of the air–cache) can be dynamically adjusted so that the system can perform as expected. For this reason, we need to establish a proper set of conditions C for the server to meet, such as maintain a certain balance between broadcast and on–demand average service time, limit the cumulative rate of explicit requests, or hold the size of the server input queue under a threshold.

We define the sets $\mathcal{V}$, $\mathcal{L}$ and $\mathcal{F}$ of vapor objects, liquid objects, and those frigid objects for which there are requests pending in the server's input queue. The server monitors the requests and estimates the temperatures for all objects in $\mathcal{V}$, $\mathcal{L}$ and $\mathcal{F}$. For objects in $\mathcal{L}$ and $\mathcal{F}$ the observed request rate is their actual request rate. Request rates for frigid objects not in $\mathcal{F}$ are assumed to be 0. This is very important since it discharges the server from keeping statistics for a large part of the database. For vapor objects however, the server observes no requests since vapor data are never explicitly requested.

Based on these estimated temperatures the adaptive broadcasting algorithm works as follows: We implement $\mathcal{V}$ as a queue and maintain $\mathcal{L}$ as an ordered list based on object temperatures. At any time the object in the head of queue $\mathcal{V}$ is scheduled to be broadcasted next. Right after broadcasted, the object is (at least temporarily) liquefied by dropping it off the queue into $\mathcal{L}$ with a reduced temperature to reflect the cooling of vapor data. Then, the server checks whether the conditions C hold. If so, then the dropped object is indeed liquefied and the size of the air–cache is reduced. If not, one or more objects from $\mathcal{L}$ are added to the end of queue $\mathcal{V}$ (they are vaporized) in decreasing order of temperature until the conditions C are met. Obviously, the size of the air–cache increases in case more than one objects are added to the queue. The object just dropped from $\mathcal{V}$ will be placed back on if its temperature is still adequately high. This leads to a repetitive (but not periodic) data broadcasting scheme since vapor objects are expected to remain hot for a while.

The estimated request rates can be exploited for buffer management at the server as well. When one or more requests cause(s) a frigid object to be read from secondary memory, the server compares its temperature to that of the coldest object in $\mathcal{L}$. If it is bigger then it "thaws" and replaces the colder liquid object which in turn freezes. In this case, the conditions C must be checked again since the new liquid object may evoke the vaporization of some liquid objects or even itself. Note that when first requested, a frigid object is appended to $\mathcal{F}$ so that its temperature gets monitored and it is removed only after all pending requests for it are serviced.

[1]We are currently investigating the effects of relaxing this assumption

4 Preliminary Experiments and Results

In order to establish the potential of adaptive data broadcasting and investigate the possible alternatives, we have built a simulation model of the proposed system. We have modeled a large client–server information system based on DBS technology where a very large number of clients (in the order of thousands) can access information either by filtering the broadcasted data stream or by explicitly requesting information from the server. For the simulation, the whole client population is modeled as a single module that imposes the total workload to the server, a stream of independent requests for objects. The rate of these requests implicitly suggests the number of clients to the system. The database is a collection of self–identifying objects each of equal size.

For the initial set of experiments we set the communication parameters to match those of the DirecPC^{TM} environment [5, 4]. We assumed a data broadcast rate of 12Mbps, and assymetric point–to–point connections with 400Kbps downstream and 19.2Kbps upstream rates. The database consists of 1000 large objects of size 100KB each to model multimedia information. The requests for data are exponentially distributed with a variable rate for different experiments. Each time the object to be requested is selected according to the Zipf distribution in order to create a skewed access pattern. However, the position and size of its "hot spot" are changing in order to produce a dynamic workload.

The preliminary experimental results are very promising and indeed demonstrate the advantages of this approach. In this paper, we briefly present some illustrative examples. The first figure shows the scalability potential of our system. As it is demonstrated in the figure, a pure data pull system breaks for workloads beyond its maximum throughput (about 200 requests per second in this case). The adaptive air–caching allows the system to accommodate efficiently much heavier workloads. For this example, the overall response time appears to increase only linearly (and not exponentially) with the request rate. This is a result of the most important performance property of our technique: *The overall response time depends only on the size of the hot spot and not the intensity of the workload.* In the best case where the size of the hot spot does not change with the total request rate, the performance of the system is expected to be the same for any number of clients.

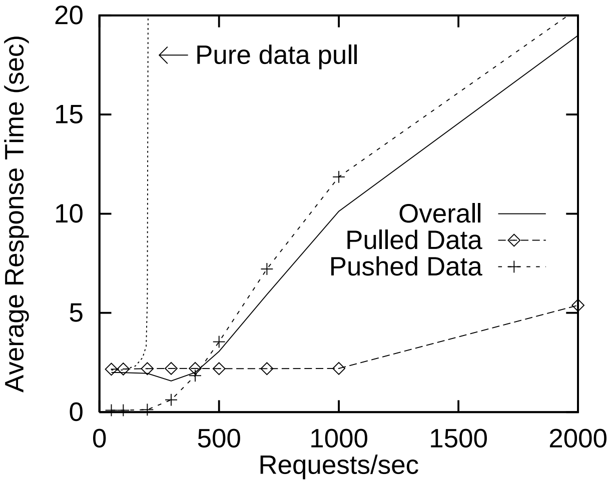

Figure 1

A related benefit of adaptive air–caching is the dramatic bandwidth savings that can be achieved. Depending on the workload, we have noticed that a pure data pull system would require hundreds more messages to be exchanged. Half of these messages would be the client requests while the other half would be the server's replies to them. In addition, the available broadcast bandwidth is very effectively utilized since, almost exclusively, only popular objects are broadcasted.

For those first experiments, we have se-

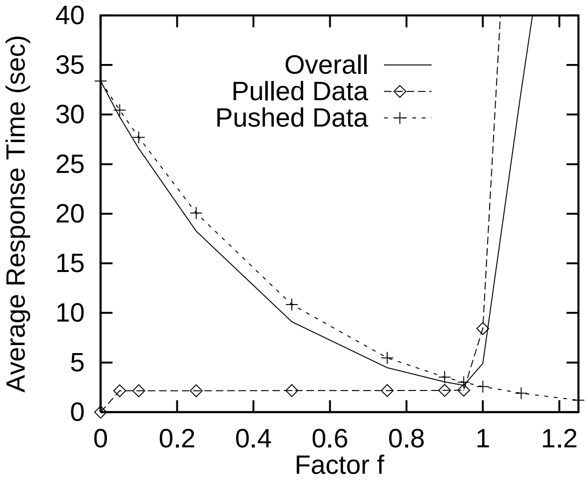

Figure 2

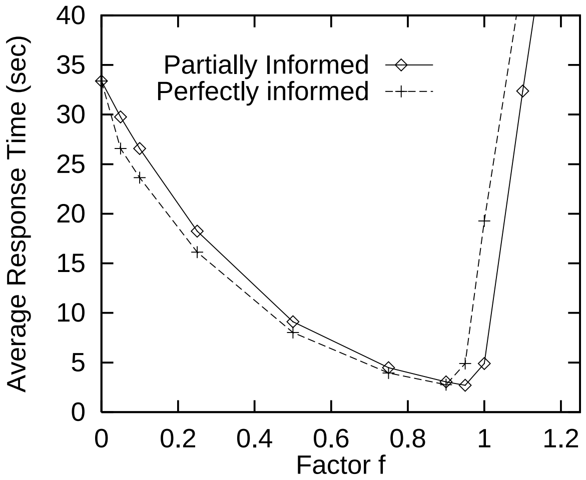

Figure 3

lected the following simple condition C: "The current rate of explicit requests must be below the server's throughput multiplied by some predetermined factor f". Let λ_V, λ_L and λ_F be the current cumulative request rates of vapor, liquid and frigid objects respectively, and μ be the maximum service rate (throughput) of the server. Note that we can consider that the current request rate for frigid objects not in $\mathcal{F}$ is 0, therefore λ_F can be approximated by the current cumulative request rate of objects in $\mathcal{F}$. Using these definitions, the condition C more formally is $\lambda_L + \lambda_F < f \times \mu$. The second figure demonstrates the advantage of the hybrid approach and the effects of the factor f. The solid line represents the overall average response time of the requests for different values of f (x axis). The shape of this line is more easily explained with the help of the other two lines which present the average response times for pushed and pulled data separately. At the left end ($f = 0$) we have a system that broadcasts all the database and the average response time depends on the size of database. At the right end (large values of f) the system chooses to broadcast very few objects, performing almost as a pure data pull system. In this case, the response time is extremely high since the workload exceeds the server's capacity. We clearly see that a hybrid approach is the best (in this case for $f \approx 0.95$).

In figure 3, the same performance result is repeated. This time we compare our server which adapts the air–cache based on partial knowledge about the usage of data to a unrealistic server that has complete knowledge (magically it knows about requests satisfied by the air–cache). We see that this lack of information makes very little difference and this is attributed to the fact that our scheme is adapting fast and usually pays a small penalty for errors. We also observe that it tends to slightly overestimate request rates and therefore broadcast a few more objects.

5 Conclusions

In this paper we have proposed adaptive air–caching as an effective way of integrating satellite communication systems with terrestrial networks to create a hybrid information system for supporting remote/mobile user applications. Our goal is to balance the need for rapid data dissemination to very large numbers of clients and on–demand interactive data services. Air–caching is based on repetitive data broadcasting and an algorithm which rapidly adapts the content of the cache based on the "misses" which result in explicit (on–demand) data requests. The initial simulation results provide clear indications of the potential of this approach. The system can efficiently accommodate huge client populations, save valuable bandwidth by broadcasting only popular information, and adapt very well to dynamic workloads despite the incomplete available information about data need.

References

[1] Swarup Acharya, Rafael Alonso, Michael Franklin, and Michael Zdonik. Broadcast Disks: Data Management for Asymmetric Communications Environments. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose, CA, May 1995.

[2] Swarup Acharya, Michael J. Franklin, and Stan Zdonik. Dissemination-based Data Delivery Using Broadcast Disks. *IEEE Personal Communications Magazine*, 2(6), December 1995.

[3] Swarup Acharya, Michael J. Franklin, and Stanley B. Zdonik. Prefetching from Broadcast Disks. In *Proceedings of the 12th International Conference on Data Engineering*, pages 276–285, New Orleans, LA, February 1996. IEEE Computer Society Press.

[4] A.D. Falk, Vivek Arora, Narin Suphasindhu, Douglas Dilon, and John S. Baras. Hybrid Internet Access. In M.S. El-Genk and R.P. Whitten, editors, *Conference on NASA Centers for Commercial Development of Space*, number 325 in AIP Conference Proceedings, pages 69–74, New York, 1995.

[5] Hughes Network Systems, Germantown, MD. *DirecPC^{TM} Hybrid Internet Technical Specification*, November 1994. Rev.1.2.

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

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

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

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

[10] Randy H. Katz and Eric A. Brewer. The Case for Wireless Overlay Networks. In *SPIE Multimedia and Networking Conference*, San Jose, CA, January 1996.

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