Adaptive Data Broadcasting in Asymmetric Communication Environments *

Wei Wang and Chinya V. Ravishankar
Department of Computer Science & Engineering
University of California, Riverside, CA 92521
{*wangw, ravi*}*@cs.ucr.edu*

Abstract

We present a new adaptive broadcast dissemination model to support flexible responses to client requests. Several features distinguish our model. First, client queries do not target individual documents, but specify the required information by attributes. Second, clients are satisfied by responses that are sufficiently close to the desired information. Finally, the server in our model solicits randomized feedback from clients to adapt its broadcast program to client needs. Our simulation results show that our model captures the interest patterns of clients more efficiently and more accurately and scales very well with the number of clients, while reducing overall client average waiting times.

1. Introduction

A communication and power asymmetry characterizes many current and emerging information delivery applications, such as news feeds and traffic information systems. Such systems are designed to deliver data from a few servers to a large number of mobile clients, but there is significantly more "downlink" bandwidth from servers to clients than in the opposite or "uplink" direction. Also, clients have low power reserves, while servers have plenty of power. Consequently, the traditional request-response or pull model, in which clients initiate information transfers from servers is inappropriate, and will not scale with the number of clients.

Given a broadcast medium, the push approach [13] can overcome some of the scalability problems in asymmetric communication environments. Servers predict client access patterns, initiate data delivery, and broadcast information to client populations. All clients with identical interests will be satisfied simultaneously. The push model has already played an important role in our daily lives (e.g., the television programs), in computing (e.g., Bellcore's Datacycle database machine [27]) and on the Internet (e.g. the CNN's Newswatch [2]).

Unfortunately, current push-based models [5, 6, 7, 16, 17, 21, 26] typically make many restrictive assumptions. Typical assumptions are that client access patterns are static [5, 6], that it suffices to have static periodic broadcast schedules [5, 6, 7, 16, 17], that server databases are small [5, 6, 7], or that prior knowledge of the access probabilities of data items in database exists [14, 28, 29]. For servers to push meaningful data to clients and maximize the overall system performance, we need adaptive and efficient scheduling schemes, so that servers can provide high-quality services and can also scale well in terms of client populations and server database sizes.

1.1. Our Contributions

We present a new data dissemination model that generates adaptive broadcast programs to satisfy as many client requests as possible while minimizing the average waiting times for clients. First, we are able to determine the client interest patterns on-line using feedback messages from a small sample of the client population. Second, as elaborated in Section 2, we increase flexibility by allowing clients to specify their requests imprecisely. The server attempts to broadcast the smallest subset of documents that matches client requests adequately, in accordance with a similarity threshold.

Our model is superior to current models in several ways. First, the server is more responsive because a single data item may satisfy many client requests. Second, the server broadcasts only a subset of data item in its database. This model reduces client waiting times and overheads for scheduling computations. We also use a randomized feedback mechanism in our model to help servers make intelligent scheduling decisions. In current models, servers change broadcast schedules in response only to complaints from unsatisfied clients. This skews the server behavior to favor unhappy clients, possibly at the expense of a silent but satisfied majority of the client population.

* This work was supported by a grant from Tata Consultancy Services, Inc.

Finally, we demonstrate the performance of our model using real-life data collection by conducting simulations. We demonstrate that our model can scale well by examining different client populations. We use Zipf distribution and uniform distribution to shape client access patterns, showing that our model can achieve good adaptability for both of them. We also compare the performance of our model with that of existing models.

The rest of the paper is organized as follows: section 2 presents the salient features of our approach. Section 3 reviews some related work. We introduce the background for our model in section 4, and briefly present the system architecture of our model in section 5. We describe our randomized feedback mechanism in section 6, and the approximate response mechanism in section 7. In section 8, we propose an objective function for optimizing our model, and generate a near-optimal broadcast program to conform to the objective function. We conduct experiments and performance evaluation in section 9. Section 10 concludes this work.

2. Approach and Rationale

Our approach has two salient features. First, we determine the interest patterns of clients explicitly, efficiently, and on-line. Second, we satisfy clients requests approximately, subject to a user-specified error tolerance.

2.1. Flexible Queries and Responses

Current models [5, 9, 15, 16, 17, 21, 26] typically require clients to explicitly specify their requests, using data item numbers or document names, for example. Unfortunately, clients may not know document names, or even what the server holds. Also, a client request can often be satisfied by any of a set of documents in the server. For example, a query for the current temperature in a city should not have to request a specific document by name (say a web page) since the client may not know the exact name of the document. It should be allowed to make a more generic request, specifying keywords such as *temperature*, and the city's name. Surely, the client will be satisfied with other documents containing this information, the home page of a local newspaper, for example. It may even be satisfied with the temperature in another city nearby.

Some methods broadcast *all* documents in the database [5, 6, 29]. This is both impractical and unnecessary. In practice, it is likely that a single document will satisfy many client requests, even if they are not identical. The home page of a local newspaper is likely to satisfy requests for weather, for news, for information regarding current events in the city, and perhaps even requests for weather in adjoining locations. In real life, requests frequently follow the Zipf distribution, so that many client requests tend to be for similar documents. In principle we can broadcast the smallest set of documents from the database that satisfies client requests given the similarity threshold. This approach minimizes average client waiting times.

Servers in our model use an approximate response mechanism to broadcast data items that are likely to satisfy client request approximately. Only data items with similarity values above a predefined similarity accuracy are assumed to satisfy client requests. This threshold is tunable, subject to parameters such as client requirements, system workloads, and so on.

2.2. Integrating Client Feedback

Some models [7, 16, 26] allow clients to make explicit requests to the server, and interleave the broadcast program and explicit requests on the broadcast medium. Unfortunately, these systems do not truly capture the interest patterns across the client population. Other models [17] have clients send explicit feedback to servers when they have unsatisfied requests. However, soliciting feedback only from unsatisfied clients will skew the server's view in their favor. Even if only a small fraction of clients are unhappy, the system will try to accommodate them, at the expense of the majority.

Since it is impossible to solicit feedback from all clients in a large population, we develop a randomized feedback mechanism (see Section 6) that serves as a random sample from the client population. At any given time, each client sends a feedback message to the server with a small probability p. The feedback is a bit vector with bit i set if the client is happy with the i-th distinct document in the broadcast. Rescheduling occurs when enough feedback has been collected, but frequently enough to reflect current client interest patterns.

3. Related Work

Much work has appeared on data dissemination recently [5, 7, 16, 17, 21, 26, 29], but several issues have yet to be adequately addressed. Some models, such as the Broadcast Disks (BD) model [5], assume static client access patterns. This pure push-based data delivery model achieves scalability by repeatedly broadcasting data items of common interest to a large client population, so clients with the same requests can be served simultaneously. However, the BD model only performs well in fairly stable environments, since its schedules do not incorporate feedback from clients effectively.

Hybrid data dissemination schemes [7, 16, 26] include a backchannel for clients to explicitly request data items not in the standard broadcast cycle for immediate delivery over

the broadcast channel. Two data transmission modes exist in such delivery schemes: the periodic broadcast or push mode and the on-demand broadcast or pull mode. Servers interleave the pushed and pulled data items based on an ad-hoc bandwidth partition parameter. The main advantage of the hybrid schemes is that servers can schedule data of common interest in the periodic broadcast program, and other data items in the on-demand pull mode, reducing average waiting times. The main disadvantage to the hybrid scheme, is that it is hard to estimate the access patterns across the client population based only on such explicit requests, since satisfied clients do not communicate with the servers.

The use of client feedback and bit vectors is not new [8, 17, 31]. It has been suggested that each client maintain its own bit vector to provide client access statistics [17]. The major problem with current methods is that clients send feedback to the server only when they are unsatisfied. Consequently, changes made by the server may be biased towards a minority of unhappy clients, at the expense of a satisfied majority. This majority would then be poorly served, triggering a massive amount of negative feedback from them.

On-line algorithms to make broadcasting data more adaptive to dynamic client access patterns have been proposed in [14, 28, 29]. However, such work focuses only on scheduling, assuming that interest patterns are already known to the server. Other models [5, 6] assume small-sized server databases. Servers schedule all data items in the databases into broadcast programs and ignore computation overheads incurred for making scheduling decisions. When access probabilities change significantly or the server database sizes increase, estimating the access probabilities of all data items in server databases will result in fairly high scheduling overhead.

Also, as discussed in Section 2, current models do not deal with the issues of flexible queries and responses. We address these issues in our new data dissemination model, which adapts to dynamic client access patterns, and scales well for larger client populations and larger database sizes. We study the problem in on-line settings.

4. The Vector Space Model

We focus on text-based documents, without loss of generality, and choose the Vector Space Model (VSM) [25], the most widely used information retrieval model, as the request-document matching model in our work. The VSM characterizes a document by the terms it contains. Terms are content-bearing words extracted from a document collection. Words in the collection are first reduced to their word stems using a well-defined set of rules [22]. Furthermore, words that are largely irrelevant to the information content of documents (such as *and, the, of, to*, and so on) are placed in a *stoplist* [12] and filtered out.

The VSM represents documents as vectors of terms in a high-dimensional vector space. Each unique term corresponds to one dimension in the space. A non-negative weight is assigned to each document along each dimension based on the term's importance within the document. Higher weights can be assigned to more important terms. The weight of a term is commonly determined based on the TF-IDF weighting scheme [24]. The *term frequency* $f_T(t_i)$ indicates the number of occurrences of a term t_i in a document. The *document frequency* $f_D(t_i)$ of a term t_i is the number of documents in the document collection that contain term t_i. The more frequently a term t_i occurs in a document, the more its importance in that document. However, if a term occurs in many documents, it may be less significant in that document collection. Therefore, we must also consider $f_D(t_i)$ in calculating a term weight.

Length normalization is also applied to documents for deemphasizing differing document lengths. It is done by dividing each document by its Euclidean length. The weight, w_i, of i-th term t_i in a document vector is

$$w_i = \frac{f_T(t_i) * \log \frac{|D|}{f_D(t_i)}}{\sqrt{\sum_{j=1}^{n} \left(f_T(t_j) * \log \frac{|D|}{f_D(t_j)}\right)^2}}$$

where n is the length of the document vector, and $|D|$ is the number of documents in collection. A document vector $\vec{d}$ is represented by (*term, weight*) pairs as $\vec{d} = \langle (t_1, w_1), \cdots, (t_n, w_n) \rangle$. Natural language requests entered by users are also converted into weighted term vectors.

4.1. Measuring Document Similarity

The angle between two vectors can be a more reliable indication of the content similarities of document vectors than the distance between them [12]. Jaccard, Dice and Cosine coefficients [23] can be used to measure the angle between a request vector and a document vector. We use the cosine coefficient measure in our study since it is the most popular similarity measure method in literature. The *cosine similarity* between a user request vector $\vec{r}$ and a document vector $\vec{d}$ is measured by a vector *inner-product* function, which can be formulated as:

$$cos\_sim(\vec{r}, \vec{d}) = \frac{\vec{r} \cdot \vec{d}}{||\vec{r}||||\vec{d}||} = \vec{r} \cdot \vec{d} = \sum_{t} \left(w_{t,\vec{r}} * w_{t,\vec{d}}\right)$$

where t is a term present in both $\vec{r}$ and $\vec{d}$. $w_{t,\vec{r}}$ is the weight of term t in $\vec{r}$ and $w_{t,\vec{d}}$ is the weight of term t in $\vec{d}$. Since $\vec{r}$ and $\vec{d}$ have been normalized by their lengths, i.e. $||\vec{r}|| = ||\vec{d}|| = 1$, the cosine similarity between them is simply their inner product. The higher the *cos_sim* value, the more similar the vectors.

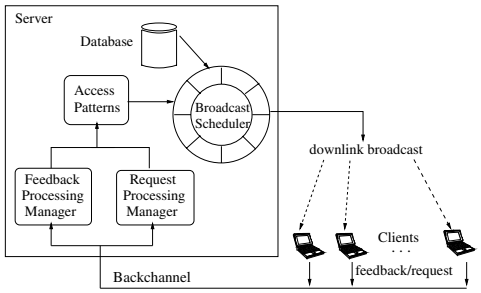


Figure 1: System Architecture

5. System Architecture

The two major components in our data dissemination model are the server and the clients, as shown in Figure 1. The server may broadcast documents relevant to various topics, such as news, stock prices, traffic conditions, weather forecast, and so on. It broadcasts a small subset of documents it holds, selected based on the random client-feedback sample, and the approximate response mechanism which we will now discuss in detail.

There are several functional units in the server. The feedback manager estimates the number of feedback and/or client explicit requests that should be collected for the server to summarize interest patterns with a desired precision (see Section 6). It monitors the incoming feedback and exploits interest patterns of the documents in the current broadcast program. The request manager deals with the sampled client explicit requests incrementally based on our approximate response mechanism (see Section 7). These two managers coordinate to help the server capture the access patterns of the entire client population. The broadcast scheduler is then used to generate new broadcast programs.

The clients listen to the broadcast and download the documents they need. At the same time, they keep evaluating the broadcast and generating feedback. All clients send feedback at random times through the backchannel. If a client is not satisfied with the broadcast, its explicit request will be taken as its feedback.

5.1. Broadcast Program Structure

The broadcast program determines the documents to be broadcast, and their order. Assume N documents in the broadcast cycle, that document d_i has size l_i, and that it takes one time unit to broadcast a document of unit length. Let the broadcast cycle broadcast documents of combined length of L units. For a skewed client access pattern, some documents will appear more than once in a broadcast cycle. Each occurrence is referred to as a copy of the document. The number of copies of a document d_i in a broadcast cycle is called its frequency and is denoted as f_i. The size of a broadcast cycle is therefore given by $\sum_{i=1}^{N} f_i l_i$. Finally, the *spacing* between two copies of a document is the time it takes to broadcast information from the beginning of the first copy to the beginning of the second copy. Figure 2 shows an example of a part of a broadcast program in our model. We can also spare a small fraction of the bandwidth for broadcasting control segments so that system performance can be tunable to the workloads.

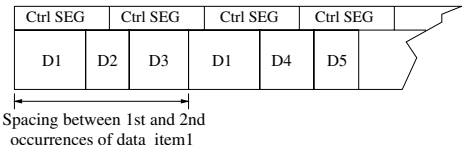


Figure 2: Part of a broadcast cycle

6. Client Interests and Randomized Feedback

Our model incorporates a mechanism for randomized feedback from client, allowing the server to estimate the client interest patterns. It integrates this information with explicit requests from clients in constructing a new broadcast schedule. In Section 6.1, we present the structure of client feedback, and in Section 6.2, we estimate how many feedback messages are required for the server to form reliable estimates of client interests.

6.1. Structure of Client Feedback

Each client continually evaluates each document in the broadcast program, and constructs a feedback vector indicating whether or not each document meets its requirements. Let r_i be the set of keywords that characterizes the client's interests. Let the similarity between client request r_i and document d_j be denoted by $cos\_sim(r_i, d_j)$, as computed in Section 4.1. We say that a client is satisfied with d_i if this similarity is no lower than a threshold τ maintained by the server and broadcast in the control segments of the broadcast program. Thus, client C_i sends the feedback bit vector $F_i = \langle f_{i1}, \ldots, f_{iN} \rangle$ to the server, where

$$f_{ij} = \begin{cases} 1 & \text{if } cos\_sim(r_i, d_j) \geq \tau \\ 0 & otherwise \end{cases}$$

Thus the feedback of C_i will be a bit string. If C_i is not satisfied with any document in the broadcast, it sends an explicit request r_i to the server as a feedback message.

6.2. Client Population and Sample Size

In wireless communication environments, it is impractical for servers to analyze feedback from all clients in a large population. The communication and computation overheads are simply too large. Our model effectively takes random samples of client feedback and explicit requests, and forms estimates of client interest based on them.

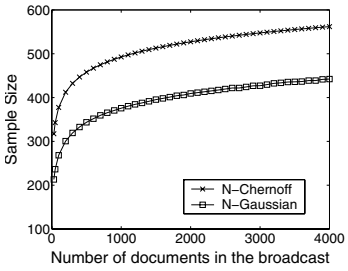


Figure 3: Required Sample Sizes Compared ($\epsilon = 0.1, \delta = 0.1$)

Let us say that we sample M clients randomly and independently in estimating the interest patterns of the entire client population. We must determine M. Let $X_1, \ldots, X_M$ be the random sample of client feedback vectors. Element X_{ij} of vector X_i is 1 if client i is satisfied with document d_j. Obviously, X_{ij} are independent since clients are chosen independently. Let N be the number of distinct documents in the broadcast. Let p_j denote the expected fraction of the client population interested in document d_j, $1 \leq j \leq N$, and let $\widehat{p_j}$ be our estimate of p_j, computed as

$$\widehat{p_j} = \frac{1}{M} \sum_{i=1}^{M} X_{ij}.$$

For each d_j in the broadcast, we want to estimate p_j within some absolute error bound ϵ. We will require

$$Pr[\,|\,\widehat{p_j} - p_j\,|\,\geq \epsilon\,] \leq \delta \qquad (1)$$

where δ is the probability that $\widehat{p_j}$ deviates from p_j by more than ϵ. In estimating p_j, $1 \leq j \leq N$ so that Equation 1 is satisfied, we will require

$$Pr[\max_{1 \leq j \leq N} |\,\widehat{p_j} - p_j\,|\,\geq \epsilon\,] \leq \delta.$$

Several statistical techniques, such as the Chebyshev [20] and Chernoff [19] bounds can be applied to determine the sample size of our estimation problem. Using the Chernoff bound, which we call it *N-Chernoff* method, we obtain a required sample size of (see [30] for details):

$$M = \frac{1}{2\,\epsilon^2} \ln \frac{2}{1 - (1-\delta)^{1/N}}.$$

In our work, we develop a new method that can yield an even tighter bound for the sample size, which we call the *N-Gaussian method*. The sample size estimated using our method is (see [30] for details):

$$M = \frac{z_\alpha^2}{4\,\epsilon^2},$$

where z_α is the z-value associated with probability α, and $\alpha = (1 + (1-\delta)^{1/N})/2$.

Figure 3 illustrates the sample sizes estimated by the *N-Chernoff* method and our *N-Gaussian* method, given $\epsilon = 0.1$ and $\delta = 0.1$. We see that the *N-Gaussian* method can always give a tighter bound for the sample size. Also, with our method, the sample size increases slowly with N. In other words, this method works well when the number of clients is large, or when the client interest pattern changes dramatically. We therefore use the N-Gaussian method.

7. Approximate Response

We introduce a mechanism for approximate responses to for improved system adaptability and scalability. The mechanism results in far fewer documents scheduled in the broadcast program, reducing the overall mean waiting time encountered by all clients. In Section 7.1, we give a high-level overview of the estimation of client access patterns. The detailed clustering method for processing explicit client requests is described in Section 7.2, and the document selection for request clusters is presented in Section 7.3.

7.1. Detection of Client Access Patterns

The server summarizes client access statistics from the randomly sampled client feedback and explicit requests, and determines the set of documents of common interests to form new broadcast schedules. The new broadcast program include some documents from the previous broadcast as well as some new documents selected in response to explicit client requests.

Algorithm 7.1 describes the details of the procedure. A set P is used to store the documents to be used in the new broadcast schedule. The document corresponding to every "1" entry in each feedback vector is incorporated into P, as are explicit document requests made by clients when they are not satisfied with any broadcast document.

Algorithm 1 Detecting client access patterns

```
1:  P ← φ
2:  for each feedback F_i = ⟨f_i1, ..., f_iN⟩ do
3:      for all j such that f_ij = 1, 1 ≤ j ≤ N do
4:          if d_j ∉ P then
5:              P ← P ∪ {d_j}
6:              set weight associated with d_j to 1
7:          else
8:              increment weight associated with d_j
9:  for each explicit client request do
10:     call explicit request clustering procedure
11: call document selection procedure
12: add the selected documents to P
13: output P
```

As explained in Section 7.2, a set of clusters is maintained to represent the distribution of client interest patterns. Each document in P is placed into an appropriate cluster based on the similarity threshold τ, whose value also decides the number of clusters. If τ is high, more clusters will be formed. Each cluster is represented by a feature vector

π. We store all feature vectors in a set S_r. Each feature vector π_k is associated with a weight c_k indicating the number of client requests that are incorporated into this cluster.

7.2. Clustering of Explicit Client Requests

Algorithm 7.2 describes the details of clustering the client explicit requests. Each client explicit request r_i is compared against all the feature vectors in set S_r. If the similarity between r_i and a feature vector is above the similarity threshold τ, the request is incorporated into the cluster represented by that feature vector. If no suitable feature vector exists, r_i forms a new cluster by itself. An adaptive parameter λ is used to adjust the feature vector of a cluster after a request is incorporated into it. To treat all the requests in a cluster equally, we set $\lambda = 1/(c_k + 1)$, where c_k is the weight associated with the feature vector.

Algorithm 2 Clustering explicit client requests

```
1:  S_r ← φ
2:  for each explicit request r_i do
3:    if S_r = φ then
4:      S_r ← S_r ∪ {r_i}
5:      set weight c_i, associated with r_i to 1
6:    else
7:      find Π = {π_k : π_k ∈ S_r, cos_sim(π_k, r_i) ≥ τ}
8:      if Π ≠ φ then
9:        for each π_k ∈ Π do
10:         π ← (1 − λ) * π_k + λ * r_i
11:         S_r ← S_r − {π_k} ∪ {π}
12:         increment weight associated with π
13:     else
14:       S_r ← S_r ∪ {r_i}
15:       set weight associated with r_i to 1
16: output S_r
```

7.3. Document Selection

Algorithm 7.3 describes the details of how to select the documents for the request clusters. When the feedback has been processed, documents from the previous broadcast to be preserved in the new broadcast schedule are already in the set P. The explicit requests in the feedback are also incorporated into appropriate clusters. We next select a document to represent each cluster, based on the similarity between the selected document and the cluster feature vector, as well as the document size. Simply selecting a document most similar to the cluster feature vector is insufficient, since choosing a large document will use up space in the broadcast cycle, and may affect it adversely. We know from Equation 4 that the minimum average waiting time is achieved when the distance between two consecutive occurrences of a document d_i is proportional to $\sqrt{l_i}$, so we use $\sqrt{l_i}$ for determining the document selection.

Algorithm 3 Selecting documents for request clusters

```
1: Input: S_r
2: for each π_i ∈ S_r do
3:   S = {d_k : cos_sim(d_k, π_i) ≥ τ}
4:   select document d ∈ S with maximum cos_sim(d,π_i) / √length(d) value
```

8. Broadcast Scheduling

In Section 8.1, we obtain the optimal mean waiting time seen by all clients. This result leads to the scheduling program in Section 8.2.

8.1. Overall Mean Waiting Time

Our performance metric is the overall average waiting time across all clients. Let N be the number of distinct documents in the broadcast cycle, n_i be the number of requests that can be satisfied by document d_i in the broadcast, t_i be the mean waiting time for d_i, and f_i be the broadcast frequency of document d_i, i.e., the number of times that d_i is broadcast in one broadcast cycle. Our objective function T depends on the document broadcast frequencies as follows:

$$T(f_1, f_2, \ldots, f_N) = \sum_{i=1}^{N} n_i t_i \Big/ \Big(\sum_{j=1}^{N} n_j\Big)$$

We assume that clients generate requests at random times. As in [5], all instances of document d_i in the broadcast cycle are equally spaced, since this yields the best performance. Let s_i be the spacing between two consecutive instances of d_i, so that the average waiting time for document d_i is $t_i = s_i/2$.

In practice, clients may time out or give up on their requests if they have to wait too long. Therefore, we require that a boundary, L, defined by client patience, constrain the length of the broadcast cycle. If document d_i is broadcast f_i times within a broadcast cycle, we have $s_i f_i = L$. We then substitute for t_i, getting

$$T(f_1, f_2, \ldots, f_N) = \sum_{i=1}^{N} \frac{n_i L}{2 f_i} \Big/ \Big(\sum_{j=1}^{N} n_j\Big)$$

If l_i is the length of document d_i, the average waiting time is then subject to the following constraint:

$$\sum_{i=1}^{N} l_i f_i \le L$$

In this case, the optimal average waiting time is obtained as (see [30] for detailed calculations)

$$T_{optimal} = \Big(\sum_{i=1}^{N} \sqrt{n_i l_i}\Big)^2 \Big/ \Big(2 \sum_{j=1}^{N} n_j\Big), \qquad (2)$$

and the broadcast frequency for each document that yields optimal mean access time is

$$f_i = \frac{\sqrt{n_i/l_i}}{\sum_{j=1}^{N} \sqrt{n_j l_j}} L \quad 1 \le i \le N. \tag{3}$$

8.2. Broadcast Scheduling Program

We noted above that the occurrences of each document should be equally spaced to archive the optimal performance, but in practice it may not always be possible to do so. We therefore provide a near-optimal solution to resolve this issue.

When the occurrences of a document d_i are equally spaced, the product of the document frequency f_i and the spacing s_i between its consecutive occurrences equals the broadcast cycle length L. From Equation 3, we obtain

$$s_i = \frac{L}{f_i} = \sqrt{\frac{l_i}{n_i}} \Big(\sum_{j=1}^{N} \sqrt{n_j l_j} \Big) \tag{4}$$

It is clear that $\sum_{j=1}^{N} \sqrt{n_j l_j}$ is a constant since the set of documents to be scheduled in the new broadcast has been determined at this point, which means that $s_i \sqrt{n_i/l_i}$ have the same value for all i. We try to preserve this characteristic in our scheduling algorithm. Thus, the document with the maximum $s_i \sqrt{n_i/l_i}$ value will always be scheduled in the broadcast. The broadcast scheduling program is given in Algorithm 8.2.

Algorithm 4 Broadcast Scheduling Program

1: $len \leftarrow 0$
2: **while** $len < L$ **do**
3: let s_i be the distance between the last occurrence of document d_i and present point of scheduling
4: find document d with maximum $s_i \cdot \sqrt{n_i/l_i}$
5: append d to the schedule
6: $len \leftarrow len + length(d)$

9. Performance Evaluation

We evaluated the performance of our model through extensive simulations on real-life data. We describe our simulation environment in Section 9.1, and demonstrate our results in Section 9.2.

9.1. Simulation Environment

We implemented our simulator using CSIM [3], and modeled a single server and multiple clients. The server broadcasts documents, collects feedback messages, detects and exploits client access patterns and makes broadcast decisions. The clients continuously generate requests and provides feedback messages to the server. The document set in our simulations is the Reuters-21578 Text Categorization Test Collection [4], which is among the most widely used resources for research in information retrieval.

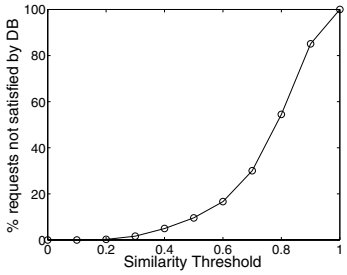


Figure 4: Client Request Characteristics

9.1.1. The Document Model. All documents in the Reuters collection are Reuters newswire stories, and belong to five different sets of content related categories, namely TOPICS, PLACES, PEOPLE, ORGS, and EXCHANGES. We used the 57 categories in the TOPICS set, obtaining 5000 documents. The documents were in SGML format, and distributed across 22 data files. We pre-processed the collection, removing SGML tags, extracting texts for each individual document, and omitting empty documents.

We removed all words in the stoplist from the documents, and reduced the rest of the words to their stems based on the PorterStemmer algorithm [1]. Finally, we converted the documents into a word-by-document matrix following the methods presented in [10]. The matrix is extremely sparse, with a density of only 0.0025. The sparse matrix was stored in the Compressed Column Storage format [11] for processing efficiency.

9.1.2. The Client Model. Each client was a CSIM process, and ran a continuous loop, with each iteration simulating one broadcast cycle. It chooses a document of interest in each broadcast cycle, and waits for a sufficiently similar document to appear in the broadcast. Each client generates (but does not send) feedback messages starting from the time it picks a document request until one broadcast cycle time elapses. If the broadcast program is changed within this time period, the client starts over on the creation of the feedback message. If no document in the broadcast cycle is sufficiently close to its document of interest, the client includes an explicit document request in the feedback message. Clients send feedback messages to the server at random times, so that the number of feedback messages arriving at the server is carefully controlled.

An explicit request vector $\vec{r}_d$ for a document is generated as a truncated version of the original document vector $\vec{d}$ by using the top five weight-ranked terms. Consequently, the similarity between the two may be less than the threshold τ. Figure 4 shows the likelihood that the database has a document matching $\vec{r}_d$ for different values of τ.

The Zipf distribution [18] is used by many current data dissemination models [5, 7, 17, 26] to model non-uniform data access patterns. For a Zipf distribution, the probability of accessing document with frequency rank r is proportional to $(1/r)^{\theta}, 0 \leq \theta \leq 1$, where θ is the access skew coefficient. If $\theta = 0$, Zipf distribution reduces to uniform distribution, and when θ increases, the client access patterns become increasingly skewed. We ran simulations for both $\theta = 0$ and $\theta = 1$, corresponding to the uniform and pure Zipf distributions. See section 9.2 for details.

To ensure that our simulations were realistic, we changed the Zipf frequencies over time, so that less popular documents became popular. Table 1 summarizes the parameters that describe the operation of the clients.

Parameter	Description	Default
θ	skew coefficient of Zipf distribution	1
ShiftFreq	shifting frequency of client access pattern	10 cycles
Offset	shift amount in client access pattern	20 docs
ReqLen	number of words of a client request	5 words

Table 1: Description of Client Parameters

The client wait time is defined as the elapsed time between request generation and its fulfillment,and it is counted in logical time units called *broadcast units*. The broadcast rate is 1KB per broadcast unit. Consequently, our simulation results are valid across many possible broadcast media. For example, if we apply our model over 2G wireless networks that have broadcast speed of 9.6Kbps, the broadcast unit would be about a second. For 2G+ wireless networks, such as GPRS, the broadcast speed is about 100Kbps, so that the broadcast unit would be about a tenth of a second.

9.1.3. The Server Model. The parameters for the server are shown in Table 2. The server is a CSIM process and runs in a continuous loop. If the server has received the required number M of feedback messages at the start of a broadcast cycle, it creates a new broadcast program as follows. First, it processes client feedback messages as in Section 7 to create feature vectors. Then it selects documents matching these feature vectors, and assigns a broadcast frequency for each document in this set based on Equation 3 in Section 8.1. Finally, it constructs the broadcast program as in Algorithm 4. In normal mode, the server listens for feedback messages from the clients. We use the CSIM event mechanism for synchronizing clients with the server.

9.2. Simulations and Results

Most current dissemination models [7, 16, 17, 21, 26] require all clients send requests to the server when they are unhappy with the broadcast cycle. These models will not scale well at the servers when client population increases or when client access patterns shift significantly. Servers in our model deal only with a sample of the entire client population, so our model scales better. We conducted extensive simulations to demonstrate the responsiveness, scalability, and adaptability of our model.

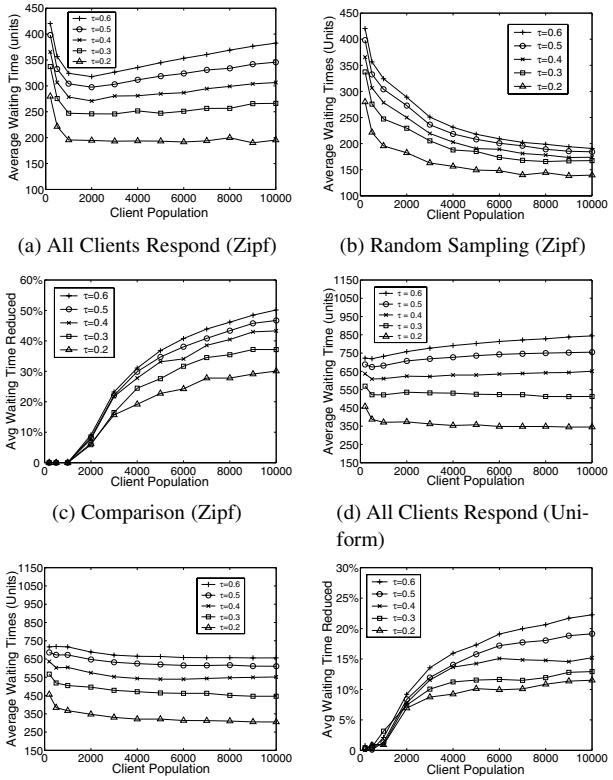
Parameter	Description	Default
L	length of one broadcast cycle	1500 units
$BRate$	broadcast rate	1KB/unit
D	size of server DB	5000
ϵ	margin of error	0.05
δ	probability of error	0.1
τ	similarity threshold	0.2-0.6
M	sample size of client feedback	
N	# of unique documents in one cycle	

Table 2: Description of Server Parameters

9.2.1. Performance Evaluation of Our Model. We compared system performance under our model and model where all clients send explicit requests to the server, varying the similarity threshold τ between 0.2 to 0.6 for both the Zipf and the uniform client access patterns. Figure 5 shows the Average Waiting Times (AWT) for different client populations. As client population increases, the AWT improves considerably for Zipf access patterns under our model, as shown in Figure 5(a) –5(c). For 10,000 clients and a similarity threshold of 0.2, the AWT is improved about 30%. For a higher similarity threshold, say 0.6, the AWT improves even more, to about 50%. The results for uniform access patterns, shown in Figures 5(d)–5(f), show similar improvements.

Figure 5 also shows that a higher similarity threshold τ leads to a longer AWT. There are two reasons for this effect. First, when τ is higher, fewer client requests are likely to be incorporated into any given cluster, so that more documents must be included in the broadcast program, leading to longer AWTs. Second, the number of explicit requests increases with τ since clients become more demanding, and are less likely to be satisfied by documents in the broadcast program. We observe that the AWT under the uniform access pattern is longer than under the Zipf pattern. Client requests are more clumped under Zipf, so that the number of documents in the broadcast program becomes smaller.

Figure 6 compares our model with existing models in terms of the percentage of unsatisfied requests in a broadcast cycle, on average. Unsatisfied requests may arise for several reasons. First, no document in the database may match a client request when τ is relatively high (see Figure 4). Second, the random sampling method in our model estimates the client access pattern with some margin of error, so that the estimate may deviate from the real pattern. Finally, the number of documents broadcast in a cycle is limited, since we limit the length of the broadcast cycle, as explained in Section 8. Figure 6(a) shows an increase in the percentage of unsatisfied requests in our model. For

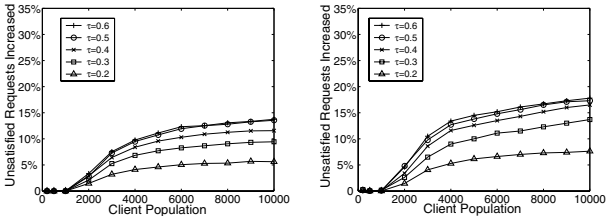


(a) All Clients Respond (Zipf) (b) Random Sampling (Zipf)

(c) Comparison (Zipf) (d) All Clients Respond (Uniform)

(e) Random Sampling (Uniform) (f) Comparison (Uniform)

Figure 5: Feedback methods compared: random sampling vs. all clients responding



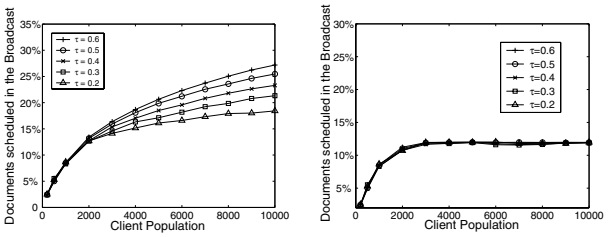
(a) Comparison (Zipf) (b) Comparison (Uniform)

Figure 6: Percentage of Unsatisfied Requests

$\tau = 0.6$, and 10,000 clients, about 13.8% client requests may be unsatisfied with the broadcast.

Figure 7 compares our method with current methods in terms of the fraction of documents scheduled in the broadcast. Our method is clearly superior, since the fraction levels off beyond a client population of 2000. The simulation results for Uniform access patterns show very similar trends. Using our model, less documents are included in the broadcast, so the broadcast frequency assigned to each document can be high, reducing the waiting times for clients interested in the document. Our method is clearly more scalable.

We also perform experiments showing how adaptable our model is under shifting client access patterns (see [30] for details).



(a) All Clients Respond (Zipf) (b) Random Sampling (Zipf)

Figure 7: Percentage of documents in server database scheduled in broadcast

Parameter	Value
θ	0.95
$ShiftFreq$	No shift
BC_Length_T	400 broadcast units
L	4000KB
$BroadcastSpeed$	10KB per broadcast unit
D	3000
$DocumentSize$	8192 bytes

Table 3: Parameter Settings

9.2.2. Comparison with Adaptive Broadcast Disk. We also compared our model with the Adaptive Broadcast Disk (Adaptive BD) scheme proposed in [17], which also explores a bit-vector feedback mechanism. In the Adaptive BD model, only clients with explicit requests will send their feedback to the server. We use exactly the same system parameter settings as those in the Adaptive BD model (see Table 3), so that their values are entirely different from those in our previous experiments. Since all documents in the Adaptive BD model have a fixed size of 8192 bytes, we ignore the actual document sizes in the database we use. Figure 8 shows our result.

Clearly, or model is much more scalable than the Adaptive BD model. In addition, the AWTs in our model are generally shorter than those in the Adaptive BD model. In the Adaptive BD model, a fixed ratio of broadcast bandwidth is allocated for broadcasting on-demand requests, so that the broadcast program can deviate from the client access patterns. The mechanisms in our model helps the server to detect and exploit client access patterns much more precisely.

10. Conclusions

In this paper, we have proposed an adaptive data dissemination model for information systems in asymmetric communication environments. We introduced an approximate response mechanism for processing client requests based on the vector space model and the cosine similarity measure. In addition, we developed a randomized client feedback mechanism, and developed a theory for bounding the sample size of client feedback using N-Gaussian method. This mechanism helps the server summarize client access patterns pre-

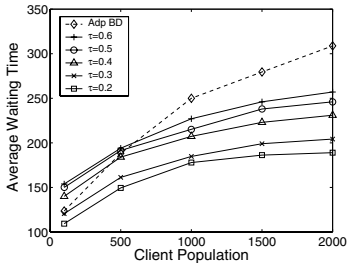


Figure 8: Performance comparison

cisely and in a timely fashion, at a very low cost. Moreover, we proposed an objective function for optimizing our model. The server creates a near-optimal broadcast program conforming to the objective function. Most importantly, all these mechanisms are seamlessly integrated into our system.

We have used real-world data set for measuring the performance of our model, using an extensive and accurate simulation testbed. Our results show that our model performs very well in terms of responsiveness, scalability and adaptability. We also compared the performance of our model with that of the Adaptive BD model in which the broadcast bandwidth allocated for explicit client request is fixed, which limits the system performance due to the dynamic nature of a data dissemination system. Our model clearly outperforms the Adaptive BD model.

References

[1] http://www.tartarus.org/˜martin/PorterStemmer/.

[2] CNN's Newswatch. http://www.cnn.com/services/newswatch.

[3] CSIM 19. http://www.mesquite.com/documentation/.

[4] Reuters-21578, v 1.0. http://www.daviddlewis.com/resources/.

[5] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communications environments. In *Proc. of the SIGMOD Conf.*, 1995.

[6] S. Acharya, M. Franklin, and S. Zdonik. Dissemination-based data delivery using broadcast disks. *IEEE Personal Communications*, 2(6), December 1995.

[7] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcast. In *Proc. of the SIGMOD Conf.*, 1997.

[8] U. Cetintemel, M. J. Franklin, and C. L. Giles. Self-adaptive user profiles for large-scale data delivery. In *Proc. of ICDE Conf.*, pages 622–633, February 2000.

[9] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive push-pull: Disseminating dynamic web data. In *Proc. of the 10th Intl WWW Conf.*, May 2001.

[10] I. S. Dhillon, J. Fan, and Y. Guan. Efficient clustering of very large document collections. In *Data Mining for Scientific and Engineering Applications*. Kluwer Academic Publishers, 2001.

[11] I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15(1):1–14, 1989.

[12] W. B. Frakes and R. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, 1992.

[13] M. Franklin and S. Zdonik. Data in your face: Push technology in perspective. In *the ACM SIGMOD Conf.*, June 1998.

[14] S. Hameed and N. H. Vaidya. Efficient algorithms for scheduling data broadcast. *ACM/Baltzer Journal of Wireless Network*, 5(3):183–193, 1999.

[15] C.-L. Hu and M.-S. Chen. Dynamic data broadcasting with traffic awareness. In *Proceedings of the 22nd ICDCS*, 2002.

[16] J.-H. Hu, K. Yeung, G. Feng, and K. Leung. A novel push-and-pull hybrid data broadcast scheme for wireless information networks. In *IEEE Int. Conf. on Communications*, volume 3, pages 1778–1782, 2000.

[17] Q. Hu, D.-L. Lee, and W.-C. Lee. Dynamic data delivery in wireless communication environments. *Workshop on Mobile Data Access*, pages 213–224, November 1998.

[18] D. Knuth. *The Art of Computer Programming, Vol II*. Addison Wesley, 1981.

[19] C. McDiarmid. On the method of bounded differences. In *Survey in Combinatorics*, pages 148–188. Cambridge University Press, 1989.

[20] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge; New York: Cambridge University Press, 1995.

[21] J. Oh, K. A. Hua, and K. Prabhakara. A new broadcasting technique for an adaptive hybrid data delivery in wireless mobile network environment. In *Proceedings of IEEE Intl. IPCCC Conf.*, pages 361–367, 2000.

[22] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[23] G. Salton. *Automatic Text Processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley publishing Company, 1988.

[24] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. New York: McGraw-Hill, 1983.

[25] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *The Communications of the ACM*, pages 613–620, November 1975.

[26] K. Stathatos, N. Roussopoulos, and J. S. Baras. Adaptive data broadcast in hybrid networks. In *Proc. of VLDB*, 1997.

[27] T.F.Bowen, G.Gopal, G.Herman, T.Hickey, K.C.Lee, W.H.Mansfield, J.Raitz, and A. Weinrib. The datacycle architecture. *Communications of the ACM*, 35(12), December 1992.

[28] N. H. Vaidya and S. Hameed. Data broadcast scheduling: On-line and off-line algorithms. Technical Report 96-017, Dept. of Computer Science, Texas A&M University, 1996.

[29] N. H. Vaidya and S. Hameed. Scheduling data broadcast in asymmetric communication environments. *Wireless Networks*, 5:171–182, 1999.

[30] W. Wang and C. V. Ravishankar. Adaptive data broadcasting in asymmetric communication environments. Technical report, University of California, Riverside, http://www.cs.ucr.edu/downloads/techrpt/TR_Adpdb.pdf, April 2004.

[31] K.-L. Wu, P. S. Yu, and M.-S. Chen. Energy-efficient caching for wireless mobile computing. In *Proc. of the ICDE*, 1996.