Adaptive Broadcasting for Similarity Queries in Wireless Content Delivery Systems

Wei Wang and Chinya V. Ravishankar, *Senior Member*, *IEEE*

Abstract—We present a new adaptive and energy-efficient broadcast model to support flexible responses to client queries. Clients do not have to request documents by name, since they may know the characteristics of the documents but not the document names or IDs. In our model, clients specify requirements through attributes, and servers broadcast documents that match client requests at a prespecified level of similarity. A given document may satisfy several clients, so the server broadcasts a minimal set of documents that achieves a desired level of satisfaction in the client population. The server obtains randomized feedback from clients and adapts its broadcast program accordingly. Clients use a selective tune-in scheme based on approximate indexing to conserve energy. Our model captures client interest patterns efficiently and accurately and scales very well with the number of clients while reducing the overall client average waiting times. The selective tune-in scheme reduces client energy consumption greatly, with a modest wait time increase.

Index Terms—Broadcast dissemination, client feedback, power management, approximate indexing, similarity matching.

1 Introduction

Mobile computing has increased the demand for online information such as stock quotes, weather, news, traffic information, and schedules at bus stands, railway stations, and airports. Shopping malls may broadcast information wirelessly about promotional sales, store hours, movie schedules, or even images of various products for the benefit of customers.

A bandwidth and power asymmetry exists in such systems. There is far more "downlink" bandwidth from servers to clients than in the "uplink" direction. Mobile clients have low power reserves, but servers have plenty of power. The push model outperforms the pull model in such applications [1], [2], [3], [4], [5], [6], [7], [8]. Pull increases traffic and resource consumption at both the client and server [1]. Push is now used in entertainment (for example, broadcasting), in computing (for example, the Datacycle DB machine [9], [10]), and on the Web (for example, the NewsDirect feed from the *Los Angeles Times* [11] and *Newswatch* from CNN [12]). In battlefields, soldiers using pull to get information from command posts [13] would reveal their positions. We will show how we can tailor broadcasts to client interest patterns, allowing push to scale well in asymmetric bandwidth environments. All clients with identical interests will be satisfied simultaneously, eliminating the need for individual responses. We will also show that push can help clients economize on battery power.

Unfortunately, current push models [5], [14], [15], [16], [17], [18], [19] typically make restrictive assumptions such as that it suffices to have static broadcast schedules [5], [14], [15], that server databases are small [5], [14], [15], that client interest patterns are static [5], [14], or that the access probabilities of data items in database are known [6], [20], [21], [22]. We need efficient and adaptive schemes for servers to provide high-quality services that scale with client numbers and server database sizes.

Current models subject clients to a "take-what-you-get" attitude on the server's part. Clients making explicit requests may get the server to broadcast what they need. The server, however, is unable to determine whether other clients are also thereby satisfied. In contrast, our approach guarantees that clients are very likely to get *at least an approximate* answer to their queries. We cannot achieve this goal by broadcasting exact responses to client requests, as other approaches do. Instead, we cluster them and broadcast documents from the server database that match the characteristics of the cluster. We determine client interest patterns explicitly, efficiently, and online. We are able to reduce the number of documents broadcast for a given level of satisfaction among the clients, significantly lowering bandwidth requirements and improving efficiency.

1.1 Flexible Queries and Responses

Existing dissemination systems [5], [16], [17], [18], [19], [23], [24] typically require clients to specify their requests by using data item numbers, document names, or explicit URLs. This model is unreasonable. First, clients may know what data they need, but not the matching document names or URLs. A client may also be satisfied with any of several documents. For example, a client looking for shirts in a mall may be happy with one of several stores. Another seeking the temperature in Chicago may be happy with the temperature in a neighboring city or with a newspaper page with this information. It is better to specify keywords such as "temperature" and "Chicago" rather than, say, a specific URL from a local weather station.

• *The authors are with the Department of Computer Science and Engineering, University of California at Riverside, Bourns Hall, Riverside, CA 92521. E-mail: {wangw, ravi}@cs.ucr.edu.*

Manuscript received 24 Oct. 2006; revised 17 July 2007; accepted 11 Oct. 2007; published online 25 Oct. 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0494-1006.
Digital Object Identifier no. 10.1109/TKDE.2007.190717.

A newspaper's homepage may satisfy queries for weather, news, current events, and, perhaps, even weather in adjoining cities. Requests frequently follow the Zipf distribution [25] so that many client requests tend to be for similar documents.

Some methods in the literature broadcast *all* documents in the database [5], [14], [20], [21]. This is impractical and unnecessary, since one document may satisfy many queries, even if they are not identical. Our model picks documents that likely satisfy many client requests, albeit approximately, using the cosine-similarity measure [26] to estimate the similarity between queries and documents. Only documents showing similarity values above some similarity threshold are assumed to satisfy client requests. This threshold is tunable, subject to client requirements, system workloads, and so on.

1.2 Integrating Client Feedback

No data delivery scheme will be robust to changing interest patterns without client interaction. Some models [15], [16], [18] allow clients to make explicit requests to the server and interleave the broadcast program and explicit requests on the broadcast medium. Other models [17] have clients sending explicit feedback to servers when they have unsatisfied requests. Unfortunately, these systems suffer from a serious flaw: *soliciting feedback solely from unsatisfied clients skews the server's view in their favor*. Even if only a small fraction of clients are unhappy, the server will try accommodating them, at the expense of the majority. These systems do not truly capture the interest patterns across the client population.

Clearly, the more client feedback the server collects, the more precisely it knows interest patterns. However, it is impossible to solicit feedback from all clients in a large population. We therefore develop a randomized feedback mechanism (see Section 3.1) 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 Poisson probability p. The feedback is a bit vector, with bit i set if the client is happy with the $i\mathrm{th}$ document in the broadcast. The server processes these feedback messages online. The randomized mechanism also balances old and new client requests by collecting enough feedback for the server to estimate client interest patterns, without waiting so long that shifts in client interest patterns occur. The server aggregates feedback, summarizes statistics, and then changes broadcast schedules accordingly.

1.3 Our Contributions

We present a new power-aware data dissemination model for asymmetric communication environments. We study ways of generating adaptive broadcast schedules that satisfy as many clients as possible while minimizing the average client waiting times. We also minimize client tune-in times to minimize power consumption. We achieve these goals by several means. First, we determine the client interest patterns online by using feedback messages from a sample of the client population. Second (see Sections 1.1 and 1.2), we increase flexibility by allowing approximate client queries. The server broadcasts the smallest subset of documents that matches client requests, in accordance with a similarity threshold. Third, we introduce a new indexing method that accommodates the flexible request and response features of our approach. Clients locate the required documents through these indices and can switch between active and sleeping modes, thereby conserving power.

Our approach has two advantages over current models. The server is more responsive, since a single data item may satisfy many client requests simultaneously. In addition, the server broadcasts only a subset of data items in its database, reducing broadcast size, client waiting times, and scheduling overheads. This is particularly useful for large server databases.

Our integration of client feedback into our scheduling model helps servers make intelligent scheduling decisions. This addresses a significant problem in existing models, in which servers change their broadcast schedules only when unsatisfied clients complain. This approach skews the server behavior to favor unhappy clients, possibly at the expense of a silent but satisfied majority of the client population.

Flexible requests and responses enable us to introduce a new indexing method that uses approximate matching to locate documents for clients. Clients can switch easily between the active mode and the sleeping mode, waking up only when the desired documents are broadcast. Client power can be conserved.

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

This paper is organized as follows: Section 2 reviews previous work. Section 3 presents our architecture and the structure of feedback. Section 3.1 describes how we can estimate sample size for our randomized feedback mechanism. Section 5 shows how we can construct the client interest patterns and proposes an objective function for optimizing our model and to generate a near-optimal broadcast program. In Section 6, we integrate an indexing scheme based on approximate matching into our model to significantly reduce client average tune-in times. We evaluate the performance of our method in Section 7. Section 8 concludes this work.

2 Related Work

Data dissemination in asymmetric bandwidth environments has been studied [5], [15], [16], [17], [18], [19], [21], but many issues have not been adequately addressed. Models such as Broadcast Disks (BD) [5] assume static client interest patterns. This model repeatedly broadcasts data items, so clients with the same requests can be served simultaneously. However, its schedules do not incorporate feedback from clients effectively, so it does well only in fairly static environments,

Some schemes [15], [16], [18] include a back channel for clients to explicitly request data items not in the broadcast cycle for immediate broadcast. Two modes exist in such schemes: a periodic broadcast (push) mode and an on-demand broadcast (pull) mode. Servers interleave data items in the periodic broadcast program with data items pulled by clients, based on an ad hoc bandwidth partition parameter. The advantage of such schemes is that they combine the push and pull models. Servers can schedule documents of common interest in the periodic broadcast program, placing others in the pull mode, and reducing the average wait times. Their main disadvantage, however, is that the true client interest patterns remain unknown, since satisfied clients are never heard from.

Clients also send feedback in [17], [27]. A bit vector is often used for delivering information compactly in asymmetric environments [28], and it has been argued that each client maintains a bit vector to collect access statistics [17]. The common problem with current methods is that the feedback represents a biased client sample, since only unhappy clients communicate with the server. Consequently, the server could be biased toward 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.

Other approaches [6], [20], [21], [29], [30], [31] assume that client interest patterns are known, or that clients explicitly pull data items from a server. In [6], [20], and [21], document popularities are assumed to be static and known. Nonetheless, documents are scheduled one at a time at the server, forcing clients to remain in the listen mode, wasting power. In contrast, we evaluate interest patterns dynamically. Our broadcast schedule minimizes the average waiting time (AWT) at clients and allows them to save power by switching to sleep mode.

The RxW scheduling algorithm [29], [30] broadcasts data items with a large number of outstanding requests, or for which the waiting times are long. RxW is unsuitable in our context. It assumes fixed-sized data items and requires clients to know and use document identifiers explicitly. We make the realistic assumption that clients do not know these identifiers, and we allow them to request documents based on a similarity metric. Variable-sized data items are considered in [31], but since preemption of broadcasts is allowed, clients cannot predict their wait times and must waste power in the listen mode. Clients pull data in [29], [30], [31], in contrast to our push-based model.

Some models [5], [14] assume small-sized server databases so that servers may schedule *all* data items into broadcast programs and ignore computation overheads incurred for making intelligent 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.

3 A New Adaptive Data Dissemination Model

The two major components of our data dissemination model are the server and the clients, as shown in Fig. 1. The server has several components. The randomized feedback manager estimates the number of feedback and explicit client requests needed to estimate interest patterns with a desired precision (see Section 3.1). The request processing manager deals with the sampled client explicit requests incrementally based on our approximate response mechanism (see Section 5). The broadcast scheduler generates new broadcast programs.

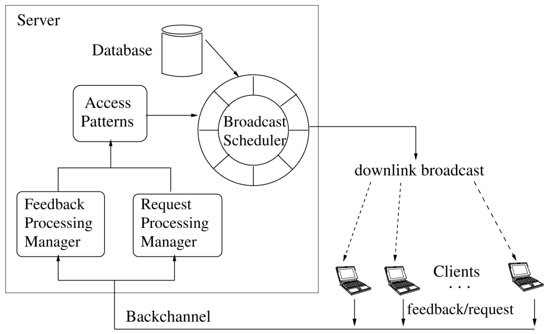


Fig. 1. System architecture.

The clients listen to the broadcast and download needed documents as they appear. They also generate a feedback vector, which they send to the server at random times through a back channel, as described in the following. If a client is unable to find a document in the broadcast that matches its needs, it sends an explicit request for the document as its feedback message.

3.1 Estimating Client Interests via Randomized Feedback

Clients generate feedback in accordance with a Poisson process with rate p, determined and broadcast by the server. A client sends a feedback message through the back channel whenever a Poisson event occurs. In Section 4, we estimate how many feedback messages are required for the server to form reliable estimates of client interests. The server estimates the Poisson parameter p based on this sample size.

A feedback message is a bit vector, with bit j set if the $j\mathrm{th}$ document in the broadcast is of interest to the client. If no broadcast document is of interest, it sends an explicit request for a document. The server considers both kinds of feedback when constructing a new broadcast schedule.

3.1.1 *Structure of Client Feedback*

Without loss of generality, we consider text documents and use the Vector Space Model (VSM) [26], a widely used information retrieval model, for matching queries with documents. The VSM represents documents as vectors of terms, which are content-bearing words extracted from a document collection, in a high-dimensional vector space. Each unique term corresponds to one dimension in the space. A nonnegative weight, commonly determined based on the TF-IDF weighting scheme [32], is assigned to each document along each dimension based on the term's importance within the document. Length normalization is also applied to documents for deemphasizing differing document lengths and is done by dividing each document by its euclidean length. The weight of the $i\mathrm{th}$ term t_i 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, $|D|$ is the number of documents in the collection, $f_T(t_i)$ is the *term frequency* indicating the number of occurrences of a term t_i in a document, and $f_D(t_i)$ is the *document frequency* indicating the number of documents in the collection containing t_i. A document vector is represented as $\vec{d} = \langle w_1, \cdots, w_n \rangle$. Natural language requests are converted into weighted term vectors.

The angle between two document vectors can be a better indication of content similarity than the distance between them [33]. The Jaccard, Dice, and Cosine coefficients [34] may be used to measure the angle between request and document vectors. We use the cosine coefficient measure, since it is the most popular similarity measure method in the literature. Given a request vector $\vec{r} = \langle r_1, r_2, \ldots, r_n \rangle$ and a document vector $\vec{d} = \langle d_1, d_2, \ldots, d_n \rangle$, their *cosine similarity* is measured as

$$cos\_sim(\vec{r}, \vec{d}) = \frac{\vec{r} \cdot \vec{d}}{\|\vec{r}\| \, \|\vec{d}\|} = \vec{r} \cdot \vec{d} = \sum_t \left(r_t d_t \right),$$

where t is a term present in both $\vec{r}$ and $\vec{d}$, r_t is the weight of term t in $\vec{r}$, and d_t is the weight of term t in $\vec{d}$. Since $\vec{r}$ and $\vec{d}$ have normalized lengths, their cosine similarity is simply their inner product. The ideal similarity metric depends upon the document collection and the client population. Its determination is beyond the scope of this paper. Work exists on expanding client request terms, looking the synonyms, supercategory terms, and so on [35], [36], [37]. These methods may be used in our context but are not part of our focus. We have used exact term matching to keep the focus on the essential features of our model.

Let r_i be the set of keywords characterizing the client's interests. We say that the client is satisfied with document d_j if $cos\_sim(r_i, d_j)$ is not 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

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

If C_i is not satisfied with any document in the broadcast, it sends the explicit request r_i to the server.

4 Estimating Client Interest Patterns

We show how to estimate client interests by using a random sample based on client feedback. We now estimate M, which is the number of clients to be sampled to form an estimate of the interest patterns of the entire client population.

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. Since the clients are chosen independently, X_{ij}, $i = 1, \ldots, M$, are independent. 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

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

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

$$\Pr\left[\, |\, \widehat{p_j} - p_j \,| \geq \epsilon \,\right] \leq \delta, \tag{2}$$

where δ is the maximum acceptable probability that $\widehat{p_j}$ deviates from p_j by more than ϵ. In estimating p_j, $1 \leq j \leq N$, so that (2) is satisfied, we will require

$$\Pr\left[\max_{1 \leq j \leq N} \{|\, \widehat{p_j} - p_j \,|\} \leq \epsilon \,\right] \geq 1 - \delta. \tag{3}$$

Chebyshev [38] and Chernoff [39] bounding can be applied to our sample size estimation problem. We prefer Chernoff bounding, since Chebyshev bound is rather weak.

4.1 The N-Chernoff Bound

The Chernoff Bound [39] works as follows: Let $X_1, \ldots, X_n$ be independent random variables, with $\Pr[X_k = 1] = p$ and $\Pr[X_k = 0] = 1 - p$ for each k. Then, for any $t > 0$

$$\Pr\left[\left| \sum_{k=1}^{n} X_k - np \right| \geq t \right] \leq 2e^{-2t^2/n}.$$

From (2), using the Chernoff bounding, we have

$$\begin{aligned} \Pr[\, |\widehat{p_j} - p_j| \geq \epsilon \,] &= \Pr[\, |M\widehat{p_j} - Mp_j| \geq M\epsilon \,] \\ &\leq 2e^{-2M^2\epsilon^2/M} = 2e^{-2\epsilon^2 M}. \end{aligned}$$

This is equivalent to

$$\Pr[\, |\widehat{p_j} - p_j| \leq \epsilon \,] \geq 1 - 2e^{-2\epsilon^2 M}.$$

To satisfy (3), we need

$$\begin{aligned} \Pr\left[\max_{1 \leq j \leq N} |\, \widehat{p_j} - p_j \,| \leq \epsilon \right] &= \Pr[\, |\, \widehat{p_j} - p_j \,| \leq \epsilon, \forall j \,] \\ &= \prod_{j=1}^{N} \Pr\left[\, |\widehat{p_j} - p_j| \leq \epsilon \,\right] \\ &\geq \left(1 - 2e^{-2\epsilon^2 M}\right)^N. \end{aligned}$$

Putting $(1 - 2e^{-2\epsilon^2 M})^N = 1 - \delta$ gives the "Chernoff" bound

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

We next present a new and novel method to obtain an even tighter bound for the sample size.

4.2 Our N-Gaussian Bound

Since X_{ij}, $i = 1, \ldots, M$, are independent, each X_{ij} corresponds to a Bernoulli trial. From (1), $M\widehat{p_j}$ is binomially distributed and may be approximated by a Gaussian for large M. If we then transform $\widehat{p_j}$ into its corresponding value

X on a standard normal, $X = \frac{\widehat{p_j} - p_j}{\sqrt{\frac{p_j(1-p_j)}{M}}}$ becomes a standard normal. We begin by defining two events:

$$E = \left\{ \frac{|\widehat{p_j} - p_j|}{\sqrt{\frac{p_j(1-p_j)}{M}}} < x, \forall j \right\}, \text{ and}$$

$$F = \left\{ \frac{|\widehat{p_j} - p_j|}{\max\limits_{1 \le k \le N} \sqrt{\frac{p_k(1-p_k)}{M}}} < x, \forall j \right\}.$$

Clearly, $E \Rightarrow F$, so that $\Pr[F] \ge \Pr[E]$. For convenience, let

$$\varepsilon = \frac{\epsilon}{\max\limits_{1 \le k \le N} \sqrt{\frac{\widehat{p_k}(1-p_k)}{M}}}. \tag{4}$$

Now, we proceed as follows:

$$\begin{aligned}
&\Pr\left[|\widehat{p_j} - p_j| \le \epsilon, \forall j\right] \\
&= \Pr\left[\frac{|\widehat{p_j} - p_j|}{\max\limits_{1 \le k \le N} \sqrt{\frac{\widehat{p_k}(1-p_k)}{M}}} \le \frac{\epsilon}{\max\limits_{1 \le k \le N} \sqrt{\frac{\widehat{p_k}(1-p_k)}{M}}}, \forall j \right] \\
&\ge \Pr\left[\frac{|\widehat{p_j} - p_j|}{\sqrt{\frac{p_j(1-p_j)}{M}}} \le \varepsilon, \ \forall j \right] \ (\text{using (4)}) \\
&= \Pr[-\varepsilon \le X \le \varepsilon].
\end{aligned}$$

We can use the cumulative normal distribution function Φ to compute the above probability. We will then have

$$\begin{aligned}
\Pr\left[\, |\widehat{p_j} - p_j| \le \epsilon, \forall j \right] &= \prod_{j=1}^{N} \Pr\left[\, |\widehat{p_j} - p_j| \le \epsilon \,\right] \\
&\ge \{\Phi(\varepsilon) - \Phi(-\varepsilon)\}^N \\
&= \{2\Phi(\varepsilon) - 1\}^N.
\end{aligned}$$

Now, since N is large, it is very likely that

$$\max_{1 \le k \le N} \sqrt{\frac{\widehat{p_k}(1-p_k)}{M}} = \frac{1}{2\sqrt{M}},$$

in which $\widehat{p_k}(1-p_k)$ attains its maximal value when $p = 1/2$. Thus,

$$\Pr\left[|\widehat{p_j} - p_j| \le \epsilon, \forall j\right] \ge \left\{2\Phi\left(2\epsilon\sqrt{M}\right) - 1\right\}^N.$$

To make this equal to $1 - \delta$, choose $2\epsilon\sqrt{M} = \Phi^{-1}\left(\frac{1+(1-\delta)^{1/N}}{2}\right)$. We thus obtain the "Gaussian" sample size:

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

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

Fig. 2 shows the sample sizes estimated by the *N-Chernoff* method and our *N-Gaussian* method for $\epsilon = 0.1$ and $\delta = 0.1$. The *N-Gaussian* method always gives a tighter bound for the sample size. In addition, with this method, the sample size increases slowly with N so that it works well when the number of clients is large or when the client interest pattern changes dramatically. We therefore use the N-Gaussian method.

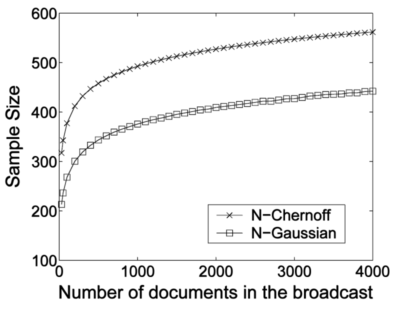


Fig. 2. Required sample sizes compared ($\epsilon = 0.1, \delta = 0.1$).

The server computes the "arrival" rate p for the Poisson process that governs the feedback generation mechanism at each client based on this sample size and places it in the control segments of the broadcast (see Section 5.2). Clients obtain p from the broadcast. When a Poisson "send" event occurs at a client, it evaluates each document in the broadcast and sends its feedback vector or explicit document request. Generally, p is very small, so feedback generation is not an onerous requirement. For example, in our simulations, the broadcast had about 600 documents per cycle, and there were 10,000 clients. The value of p works out to about 0.035.

5 Broadcast Scheduling Using Feedback

The server estimates the relative popularities of its documents based on client feedback. It clusters client queries and schedules documents to satisfy the maximum number of clients while reducing the mean client waiting times.

5.1 Using Client Feedback

The server determines the new set of documents to broadcast by constructing the distribution of client interests from the feedback vectors and explicit requests from clients. The feedback messages serve as a fair random sample from the client population. Using feedback only from unhappy clients skews the broadcast toward this client subset, at the expense of the rest (see Section 1.2). The strength of our method is that we are able to satisfy a large client population without many explicit requests. The new broadcast program includes some documents from the previous broadcast and some new documents selected in response to explicit client requests.

Algorithm 1 describes the details of the procedure. The set P holds the documents to be used in the new broadcast schedule. Each document d_j is associated with a "weight" c_j, which is a counter initialized to 0. The document corresponding to every "1" entry in each feedback vector is incorporated into P, and its corresponding weight is incremented by 1. Hence, each document d_j of continuing interest to clients will be a candidate for inclusion in the new broadcast, but its frequency will be adjusted based on the new weight c_j associated with d_j. New documents may appear in the broadcast in response to the explicit requests from unhappy clients.

Algorithm 1: Finding client access patterns

```
1:  $P \leftarrow \phi$, $c_j = 0$ for all $j$, $1 \leq j \leq N$
2:  for each feedback $F_i = \langle f_{i1}, \dots, f_{iN} \rangle$ do
3:    for all $j$ such that $f_{ij} = 1$, $1 \leq j \leq N$ do
4:      if $d_j \notin P$ then
5:        $P \leftarrow P \cup \{d_j\}$
6:        set weight $c_j$ associated with $d_j$ to 1
7:      else
8:        increment $c_j$ by 1
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 $P$
```

5.1.1 *Clustering of Explicit Client Requests*

We perform online clustering of client requests by using a method similar to that in [27]. As discussed in [27], such online clustering is more time- and cost-efficient than the traditional cluster-based algorithms, which are typically expensive and require all data to be available. This is impractical in our environment.

Explicit client requests are clustered as in Algorithm 2. The set S_r holds clusters, which are represented by feature vectors. S_r is initially empty. The clusters are formed incrementally as the server processes incoming explicit requests. Each feature vector π_k is associated with a weight c_k indicating the number of client requests incorporated into this cluster. Each explicit client request r_i is compared against the feature vectors in set S_r. If the similarity between r_i and a feature vector is above 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 $\lambda = 1/(c_k + 1)$. The value of τ also influences the number of clusters. If τ is high, more clusters will be formed.

Algorithm 2: Clustering explicit requests

```
1:  $S_r \leftarrow \phi$
2:  for each explicit request $r_i$ do
3:    if $S_r = \phi$ then
4:      $S_r \leftarrow S_r \cup \{r_i\}$
5:      set weight $c_i$ associated with $r_i$ to 1
6:    else
7:      find $\Pi = \{\pi_k : \pi_k \in S_r, cos\_sim(\pi_k, r_i) \geq \tau\}$
8:      if $\Pi \neq \phi$ then
9:        for each $\pi_k \in \Pi$ do
10:         $\pi \leftarrow (1 - \lambda) * \pi_k + \lambda * r_i$
11:         $S_r \leftarrow S_r - \{\pi_k\} \cup \{\pi\}$
12:         increment weight associated with $\pi$
13:     else
14:       $S_r \leftarrow S_r \cup \{r_i\}$
15:       set weight associated with $r_i$ to 1
16: output $S_r$
```

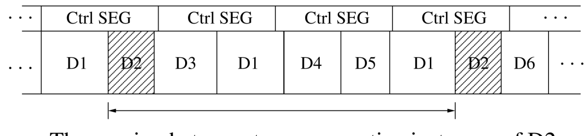


Fig. 3. Part of a broadcast cycle.

5.1.2 *Document Selection*

Algorithm 3 shows how we can select representative documents for request clusters. After the feedback has been processed, documents to be carried over into the new broadcast schedule are in the set P. Explicit requests are also incorporated into appropriate clusters. A document is selected to represent each cluster based on the similarity between this document and the cluster feature vector, as well as the document size. Simply selecting a document that is 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 (10) that the AWT is minimized when the distance between two consecutive occurrences of a document d_i is proportional to $\sqrt{l_i}$, where l_i is the length of d_i, so we use $\sqrt{l_i}$ for determining the document selection.

Algorithm 3: Selecting documents to represent clusters.

```
1: Input: $S_r$
2: for each $\pi_i \in S_r$ do
3:   $S = \{d_k : cos\_sim(d_k, \pi_i) \geq \tau\}$
4:   select document $d \in S$ with the maximum $\frac{cos\_sim(d,\pi_i)}{\sqrt{length(d)}}$
     value
```

The document selection procedure need not be run frequently when making a new broadcast schedule, unless client interest patterns change significantly, and most of the documents in the previous broadcast program are to be replaced. It needs to be executed only when there are explicit client requests, which are not likely to be many. Our clustering method can further reduce the number of documents selected.

5.2 The Broadcast Scheduler

The broadcast scheduler determines the documents to be broadcast and their order in the broadcast program. If N distinct documents must be scheduled, each must appear at least once in a broadcast cycle. The more popular documents will appear more frequently, but even the least popular document must be broadcast once in the broadcast cycle. The number of times that a document d_i is broadcast in one broadcast cycle is called its frequency and is denoted by f_i. If l_i is the length of document d_i, the size of a broadcast cycle is therefore given by $\sum_{i=1}^{N} f_i l_i$. Finally, the *spacing* between two consecutive instances of a document d_i is the time from the beginning of one instance of d_i to the beginning of its next instance. Fig. 3 shows an example of a part of a broadcast program in our model. We allocate a small fraction of the bandwidth for broadcasting control segments so that system performance can be tuned to the workload.

5.2.1 Overall Mean Waiting Time

Our performance metric is the overall AWT across all clients. We derive the optimal AWT based on the idea of satisfying all client requests within one broadcast cycle. As before, N is the number of distinct documents in the broadcast cycle, and f_i is the broadcast frequency of document d_i. Let n_i be the number of requests that can be satisfied by document d_i in the broadcast and t_i be the mean waiting time for d_i. 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 [40]. Let s_i be the spacing between two consecutive instances of d_i so that the AWT t_i for document d_i is $s_i/2$. Our objective function is the AWT, defined as

$$T(f_1, f_2, \dots, f_N) = \frac{\sum_{i=1}^N n_i t_i}{\sum_{j=1}^N n_j} = \frac{1}{2}\left(\sum_{i=1}^N n_i s_i\right) \Big/ \left(\sum_{j=1}^N n_j\right).$$

In practice, clients may time out or give up 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 $s_i f_i = L$. Substituting for s_i,

$$T(f_1, f_2, \dots, f_N) = \left(\sum_{i=1}^N \frac{n_i L}{2 f_i}\right) \Big/ \left(\sum_{j=1}^N n_j\right). \tag{5}$$

As l_i is the length of document d_i, the AWT is then subject to the following constraint:

$$\sum_{i=1}^N f_i l_i \le L. \tag{6}$$

We now have a nonlinear optimization problem cast in terms of the objective function (5), which computes the overall AWT across all clients. We apply *Lagrange's method of undetermined multipliers* [41] to minimize this function, subject to constraint (6). Multiplying (6) with an undetermined parameter λ, we have

$$\lambda\left(\sum_{i=1}^N l_i f_i - L\right) = 0.$$

We can then rewrite our objective function as

$$T(f_1, f_2, \dots, f_N) = \left(\sum_{i=1}^N \frac{n_i L}{2 f_i} \Big/ \sum_{j=1}^N n_j\right) + \lambda\left(\sum_{i=1}^N l_i f_i - L\right). \tag{7}$$

We find the minimum in (7) by differentiating the function with respect to each f_i $(i = 1, 2, \dots, N)$ and setting all the derivatives to zero. This yields

$$\frac{\partial T}{\partial f_i} = \frac{n_i L}{2S}(-1)\frac{1}{f_i^2} + \lambda l_i = 0,$$

which implies that

$$f_i = \sqrt{\frac{n_i L}{2 l_i S}}\,\frac{1}{\sqrt{\lambda}}, \quad \text{where } S = \sum_{j=1}^N n_j. \tag{8}$$

Substituting f_i in (6) with (8), we have

$$\sum_{i=1}^N l_i \sqrt{\frac{n_i L}{2 l_i S}}\,\frac{1}{\sqrt{\lambda}} = L, \quad \text{yielding} \quad \sqrt{\lambda} = \sum_{i=1}^N \sqrt{\frac{n_i l_i}{2LS}}.$$

We then calculate the stationary points of the objective function $T(f_1, f_2, \dots, f_N)$ by substituting $\sqrt{\lambda}$ into (8):

$$f_i = \frac{\sqrt{n_i/l_i}}{\sum_{j=1}^N \sqrt{n_j l_j}} L. \tag{9}$$

Substituting f_i into (5) and simplifying yields the optimal AWT as

$$T_{optimal} = \frac{1}{2S}\sum_{i=1}^N n_i L \frac{\sum_{j=1}^N \sqrt{n_j l_j}}{L\sqrt{n_i/l_i}} = \frac{\left(\sum_{i=1}^N \sqrt{n_i l_i}\right)^2}{2\sum_{j=1}^N n_j}.$$

5.2.2 Our Scheduling Algorithm

We have noted that optimal performance results when the instances of any document are equally spaced, but such equal spacing may sometimes be difficult in practice. In addition, since we bound the maximum broadcast cycle length, we may end up with computed broadcast frequencies f_i below 1 for some low-popularity documents so that the server may not be able to schedule every selected document in each broadcast cycle. However, we observe that our derived optimal AWT $T_{optimal}$ follows the square root property presented in [6] and [42]. An algorithm for achieving near-optimal performance was presented in [6], which we use as the basis for our broadcast scheduling program to address the issues of unequal spacing and $f_i < 1$.

When the instances 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 (9), we obtain

$$s_i = \frac{L}{f_i} = \sqrt{\frac{l_i}{n_i}}\left(\sum_{j=1}^N \sqrt{n_j l_j}\right).$$

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 preserving this characteristic in the scheduling algorithm. Thus, for documents with a broadcast frequency of at least 1, the document with the maximum $s_i\sqrt{n_i/l_i}$ value will always be scheduled in the broadcast. However, it may take several broadcast cycles to broadcast an instance of a document with a broadcast frequency below 1.

6 Selective Tune-in for Energy Conservation

So far, we have required clients to monitor the broadcast constantly for the documents that they require. However, listening to wireless broadcasts requires significant power

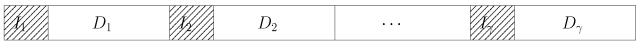


Fig. 4. Index and data segments.

[43], and clients may have low energy reserves. We now show how we can integrate an indexing scheme into our model. Clients can use the index to be selective about when to listen to the broadcast.

Devices can be in the active mode or in the doze mode, so the total energy consumption is

$$E_{Total} = E_D + E_A = E_D + (E_L + E_C),$$

where clients consume energy E_D in the doze mode and energy E_A in the active mode. E_A has two components: E_L, which is the energy consumed in the "listen" mode while accessing the broadcast, listening to the index segments, and downloading the documents, and E_C, which is the energy consumed while computing. E_D is negligible compared to E_A, since the energy consumed in the active mode is significantly higher than in the doze mode [44]. For example, an IEEE 802.11 WLAN card with a rate of 1 megabit per second (Mbps) consumes 1.4 W in the listen mode and only 0.045 W in the doze mode. Therefore, E_{Total} can be approximated as E_A. Significant energy savings can be achieved by clients selectively tuning into the broadcast using index segments and reducing the time in the active mode. The energy consumed by switching between the active mode and the doze mode is usually negligible [45].

Clients listen to the full broadcast only when sending feedback, which they do rarely, since the associated Poisson probability is small (see Section 4). At all other times, they use the index to locate the position in the broadcast of the document of interest, move into doze mode, and wake up only when the desired document is being broadcast. We have already optimized our scheme to minimize the waiting times, so adding indices will increase waiting times. An application must hence balance power savings from using the index against the increase in waiting times.

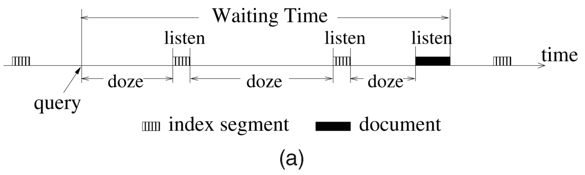
6.1 Broadcast Organization

We partition our broadcast cycle into γ data segments D_i, $i = 1, \ldots, \gamma$, and place an index segment I_i ahead of each data segment D_i (see Fig. 4). Each index segment indexes only documents appearing in the following data segment. The broadcast cycle now includes both data segments and index segments. Since the documents in the broadcast vary in size, the index segments cannot always be exactly equally spaced. For simplicity, we assume that the distances between any two consecutive index segments are nearly the same.

In addition to index and data segments, very small indicators are interleaved into the broadcast, giving the position of the next index segment. These indicators appear frequently in the broadcast, so clients can quickly locate the position of the next index segment by listening to the broadcast for a very short time. For simplicity, we have omitted indicators in Fig. 4.

6.2 Using Index Segments

Index segments are generally much shorter than broadcast length, so clients can greatly reduce the time spent in the



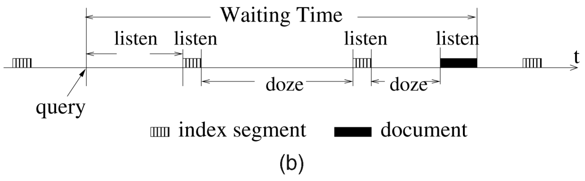


Fig. 5. The data access flow. (a) Clients listen to the index only to locate a document. (b) Clients listen to the broadcast and index to find the document.

active mode by locating required documents using index segments. We consider two methods for clients to use index segments.

In the first method (see Fig. 5a), a client uses the indicator segments to obtain the location of the next index segment, and sleeps until that index segment arrives. It then reads the index to determine if the following data segment contains the desired document and switches to the doze mode. If the document appears in the following data segment, the client uses the information in the index to wake up at the appropriate time. Otherwise, the client wakes up when the next index segment arrives.

This method has the drawback that clients may miss some relevant documents during the first doze period, that is, between the arrivals of the request and the first available index segment. This can needlessly increase waiting times.

In the second method, the client simply listens to the broadcast until the first index segment arrives. If the required document is found before the next index segment, the client is satisfied. Otherwise, it locates the required documents by listening to the index segments only, as in the first method. Fig. 5b shows the data access flow for this case. If the client obtains a satisfactory document when it listens to the broadcast, the client waiting time will be the minimum. The second method requires that clients stay in the active mode longer, but this may be more suitable for clients sensitive to waiting times.

6.3 Indices for Similarity Matching

Some indexing techniques for data broadcasting in wireless environments have appeared in the literature [46], [47], [48], [49], [50], [51], but none of these is particularly useful with similarity-based querying. In [46] and [47], a tree-structured index is created for data frames in the broadcast cycle. However, all queries are based on a single primary key, and the data frames are sorted by the primary key. The indexing approach proposed in [50] also indexes data items according to a simple identifier and does not adapt to the dynamic system workloads. Huang and Peng [51] propose an indexing method that accounts for power consumption, but this approach requires statistical information on served requests to tune the index to system workloads. Indices in all of these methods assume that the data is specified using a unique key, which is indexed. Such methods cannot

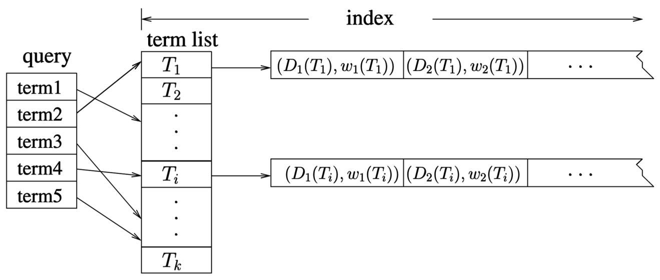


Fig. 6. The indexing structure.

support the class of queries of interest to us, in which clients do not request documents by name but rather specify them imprecisely by using multiple keywords.

Signature schemes are proposed in [48]. A signature is formed for a record by first hashing each value in the record into a bit string and combining these bit strings to obtain a record signature. A query is constructed similarly and is compared to the record signatures by performing a bitwise AND operation. This scheme introduces *false drops*, in which the signature comparison indicates a match, but in fact, the record does not match the query and must therefore be dropped. To handle false drops, each signature match must be confirmed by comparing the record with the query, which consumes more energy. A hybrid index technique that combines the strengths of the signature and the index tree techniques is proposed in [49]. In this scheme, all records must be checked when a match occurs, resulting in extra energy consumption. Again, such signature schemes are based on an exact-match paradigm, where the user specifies the required record exactly. In contrast, we allow queries to be satisfied approximately under some similarity threshold.

Our flexible request-response features require us to allow approximate matching within our indexing scheme. Our clients do not specify documents with a single key but by using attributes. Our index allows matching on attributes.

We organize the index segment I_i as an inverted term-document list (see Fig. 6). The list of terms is constructed by selecting the 10 top-ranked terms based on their weights from each document in the data segment. Terms are sorted alphabetically. Each term T_k in the list functions as a key, after which follows a list L_k of pairs of the form $(D_i(T_k), w_i(T_k))$, where $D_i(T_k)$ is the identifier of a document containing T_k, and $w_i(T_k)$ is the weight of term T_k in that document.

A query has form $Q = \{(T_1, u_1), (T_2, u_2), \dots, (T_p, u_p)\}$, where T_j represents a term of interest, and u_j is the desired weight of the term. For each query term T_j, the client can use the index to find L_j set of documents that contain T_j. The client finds $L_1, L_2, \dots, L_p$ in succession, accumulating the sum $W_i = \sum_{j=1}^{p} w_i(T_j) * u_j$ for each document $D_i(T_j)$ in these lists. Finally, the client locates a document matching its query by choosing the nearest document in the broadcast whose W_i value exceeds the predefined similarity threshold.

The term selection is a truncated version of the document, so the result of a simple comparison between a query and the term selection may be different from that of a match between the query and the full document. However, our experiments suggest that this happens rarely. The 10 top-ranked terms of a document are dominant in the whole document. We picked document pairs randomly from the database and computed the similarity s_a between them by using all their terms. We also computed the similarity s_p between them by only using the top-ranked terms. The error is, on average, 0.023, which is very small. In addition, documents in the broadcast are selected based on client interest patterns, so most clients are likely satisfied with some document in the broadcast. Therefore, we can use a lower similarity threshold when finding matches between the query and the term selections. Moreover, if a client finds no matching document, it can listen to the full broadcast.

6.4 Performance Analysis

We evaluate the data access efficiency of broadcasting with indices under the following performance metrics, which are also widely used in the literature [46], [47], [48], [49], [50], [51]:

- *Waiting time.* This is the time between query issue and document receipt at the client.
- *Tuning time.* This is the time that a client is in the active mode during the waiting time. This metric [46] evaluates the effects of indexing on energy savings.

6.4.1 Average Waiting Time

The *AWT* is measured as the average time to get to the next index segment plus the average time from the index segment to the point when the required document is downloaded.

We use the following notations: The combined size of all the data segments is L, which is the length of the broadcast cycle without index segments. $|I|$ denotes the size of a single index segment: All index segments are of the same size. The length of a broadcast cycle with γ index segments is $L + \gamma|I|$.

Since we assume near-equal placing of the index segments, the data segments sizes will all be nearly equal to L/γ. The average time to get to the next index segment is half the time between two consecutive index segments, that is, $\frac{1}{2}\left(|I| + L/\gamma\right)$. The average duration from the index segment to document d_i is half the spacing between two consecutive occurrences of d_i, which is now $(L + \gamma|I|)/f_i$. Hence,

$$AWT = \frac{1}{2}\left(|I| + \frac{L}{\gamma}\right) + \frac{1}{2N}\sum_{i=1}^{N}\frac{L + \gamma\,|I|}{f_i}.$$

We find the minimum AWT by differentiating with respect to γ and setting the derivative to zero. That is,

$$\frac{\partial(AWT)}{\partial\gamma} = -\frac{L}{2\gamma^2} + \frac{|I|}{2N}\sum_{i=1}^{N}\frac{1}{f_i} = 0, \text{ so that}$$

$$\gamma = \sqrt{\frac{LN}{|I|\left(\sum_{i=1}^{N}\frac{1}{f_i}\right)}}.$$

6.4.2 Average Tuning Time

The average tuning time (ATT) without indexing equals the AWT. The ATT with indexing is the sum of the time to get the next index segment, the time to get to the document, and the time to download the document.

The client obtains lists $L_1, L_2, \ldots, L_p$ from the index structure by using terms in its query and computes $\sum_{j=1}^{p} w_i(T_j) \times u_j$ for each $D_i(T_j)$ in these lists. This computation can be sped up using a hash table, using document identifiers as the hash key. As the client examines each $(D_i(T_j), w_i(T_j))$ tuple in L_j, it simply updates the corresponding total, incrementing it by $w_i(T_j) * u_j$. The computation overhead is linear to the number of $(D_i(T_j), w_i(T_j))$ tuples across the L_j.

We see in Fig. 11 that about 12 percent of the documents in the server database will be scheduled in the broadcast program on the average. Given a server with 5,000 documents, about 600 documents will appear in the broadcast. As we partition the broadcast cycle into $\gamma\,(\approx 10)$ data segments, there will be around 60 distinct documents in each data segment. If we assume five terms in each query, the number of $(D_i(T_j), w_i(T_j))$ tuples examined is around 300 in the worst case. Our simulations suggest that the number of $(D_i(T_j), w_i(T_j))$ tuples in each L_j is indeed very small and is only about 2 on the average. The variance in this number is also small, being around 6.23. We therefore need to search only a few tuples, on the average, and our indexing approach is efficient.

7 Performance Evaluation

We evaluated our model through extensive simulations on real-life data such as Reuters newswire text documents.

7.1 Simulation Environment

We implemented a simulator using CSIM [52] and modeled a single server and multiple clients. The server broadcasts documents, collects feedback messages, detects client interest patterns, and makes broadcast decisions. The clients generate document requests and provide feedback messages. The document set in our simulations is the Reuters-21578 Text Categorization Test Collection [53], which is among the most widely used resources for research in information retrieval.

7.1.1 The Document Model

The documents in the Reuters collection are newswire stories belonging to five different content-related categories: TOPICS, PLACES, PEOPLE, ORGS, and EXCHANGES. We used 57 categories in the TOPICS set, obtaining 5,000 documents, all in SGML format and distributed across 22 data files. We preprocessed the collection, removing SGML tags, extracting texts for each individual document, and omitting empty documents.

We removed all words in the stop list, reduced the rest to their stems using PorterStemmer [54], and converted the documents into a word-by-document matrix, as in [55]. The VSM for the 5,000 documents contained 21,485 unique terms. There were 251,475 nonzero entries in the matrix, which were term weights calculated by the TF-IDF weighting scheme (see Section 3.1.1). Each document contained about 51 terms on average. Thus, the matrix is extremely sparse, with a density of only 0.0025. The sparse matrix was stored in the Compressed Column Storage format [56] for efficiency.

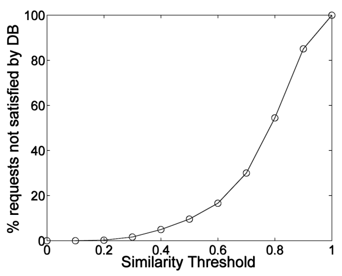


Fig. 7. Client request characteristics.

7.1.2 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 that 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, following the same Poisson distribution for all clients. The number of feedback messages arriving at the server is carefully controlled.

Explicit requests for documents are generated as follows: A document d is selected from the database according to a specified distribution, and all terms in the selected document are sorted in nonincreasing weight order. The client forms an explicit request vector $\vec{r}_d$ for d by using the five top-ranked terms in d so that $\vec{r}_d$ is a truncated version of the original document vector $\vec{d}$. Hence, the similarity between the two may be less than the threshold τ. Fig. 7 shows the likelihood that the database has a document matching $\vec{r}_d$ for different τ values.

In practice, client interests are very likely to change over time. For instance, traffic information may be more requested during commute hours, but movie schedules or shopping information may be more accessed during the day. Client request patterns have been studied in various contexts, and the Zipf distribution [57] has been found to be a good characterization of these patterns [25]. Zipf is a power-law distribution and is frequently seen, because all objects are not equally interesting to clients in any context. Zipf is widely used in many dissemination and access models [5], [15], [16], [17] and is even seen in the distribution of outdegrees in Internet routers [58]. For a Zipf distribution, the probability of accessing a document with frequency rank r is proportional to $(1/r)^{\theta}$, $0 \leq \theta \leq 1$, where θ is the access skew coefficient. If $\theta = 0$, the Zipf distribution reduces to the uniform distribution but

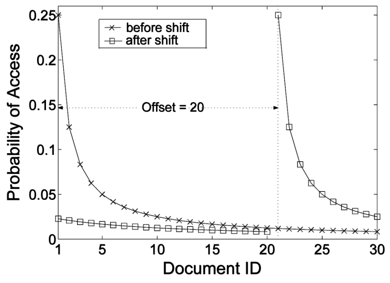


Fig. 8. Shifting client access patterns.

becomes increasingly skewed as θ increases. We ran simulations for both $\theta = 0$ and $\theta = 1$, corresponding to the uniform and pure Zipf distributions. The details are in Section 7.2.

We modeled changes in client access patterns, as shown in Fig. 8. A similar method was used in [5] and [14]. Fig. 8 assumes a database of 30 documents and shows the effects of client access patterns shifting by an offset of 20 documents. Initially, document 1 has the highest probability of being accessed, and document 30 has the lowest. After a shift in interest patterns, document 21 is the most popular document, and document 20 is the least popular. Table 1 summarizes the parameters describing client operation. *ShiftFreq* specifies how frequently the client interest patterns shift. *Offset* defines the shift amount.

The client waiting time is the time elapsed between request generation and its fulfillment. We simulate the arrival of client requests as a Poisson process. The client searches the current broadcast program for a document that satisfies a generated request. If no such document is found within one broadcast cycle, the request is flagged as pending and is processed again in the next broadcast cycle. If it cannot be satisfied in two consecutive cycles, it is considered as an unsatisfied request.

We count the waiting time in logical time units called *broadcast units*, so our simulation results are valid across many possible broadcast media. We used a broadcast rate of 1 Kbyte per broadcast unit, but we can apply our model over 2G wireless networks that have broadcast speed of 9.6 kilobits per second (Kbps) by changing the broadcast unit to a second. For 2G+ wireless networks such as GPRS, the broadcast speed is about 100 Kbps so that the broadcast unit would be about a tenth of a second.

7.1.3 The Server Model

The server parameters are shown in Table 1. The server is a CSIM process running in a continuous loop. The server collects the required number M of feedback messages and creates a new broadcast program as follows: First, it processes client feedback messages (see Section 5) to create feature vectors. It then selects documents matching these feature vectors and assigns a broadcast frequency for each document based on (9). Finally, it constructs the broadcast program as in Section 5.2.2. We use the CSIM event mechanism to synchronize clients with the server.

7.2 Simulations and Results

Current models [15], [16], [17], [18], [19] typically require unhappy clients to send explicit document requests to the server. Consequently, the server likely performs poorly as the number of clients increases or when client interest patterns shift significantly. Servers in our model deal only with a sample of the entire client population, so our model scales much better. We conducted extensive simulations to demonstrate the responsiveness, scalability, adaptability, and energy efficiency of our model.

TABLE 1
Description of Parameters

Parameter	Description	Default Value
θ	skew coefficient of Zipf distr.	1
ShiftFreq	freq. of client interest shift	10 cycles
Offset	shift in client interest pattern	20 documents
ReqLen	number of words per request	5 words
L	length of a 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	#unique docs in broadcast cycle	

7.2.1 Performance of Randomized Feedback Scheme

We measured system performance under our random feedback method, and when all clients send feedback, varying the similarity threshold τ between 0.2 to 0.6 for both the Zipf and the uniform client interest patterns. No indices were included in the broadcast in these experiments, and the server minimized the number of documents broadcast based on similarity matching. Fig. 9 shows the AWTs for different client populations. The results for the Zipf interest patterns are shown in Figs. 9a, 9b, and 9c. As client population increases, the AWT improves considerably under our model, as shown in Fig. 9c. For 10,000 clients and a similarity threshold of 0.2, the AWT is improved about 30 percent compared to the case when all clients send explicit requests to the server. For a higher similarity threshold, say 0.6, the AWT improves even more, that is, to about 50 percent. The results for uniform interest patterns, as shown in Figs. 9d, 9e, and 9f, show similar improvements. AWTs are longer when all clients respond, because the server must schedule more documents in the broadcast to satisfy all arriving client requests. One might expect that scheduling more documents would satisfy more client requests, reducing waiting times. However, given a fixed-length broadcast cycle, scheduling more documents decreases the broadcast frequencies of the documents, actually increasing waiting times. In this case, however, the server broadcasts some documents that only a small fraction of clients want, further increasing AWT.

In Fig. 9, a higher τ leads to a longer AWT for two reasons. First, for higher τ, fewer client requests are likely incorporated into a 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 satisfied by documents in the current broadcast program. We observe that the AWT under the

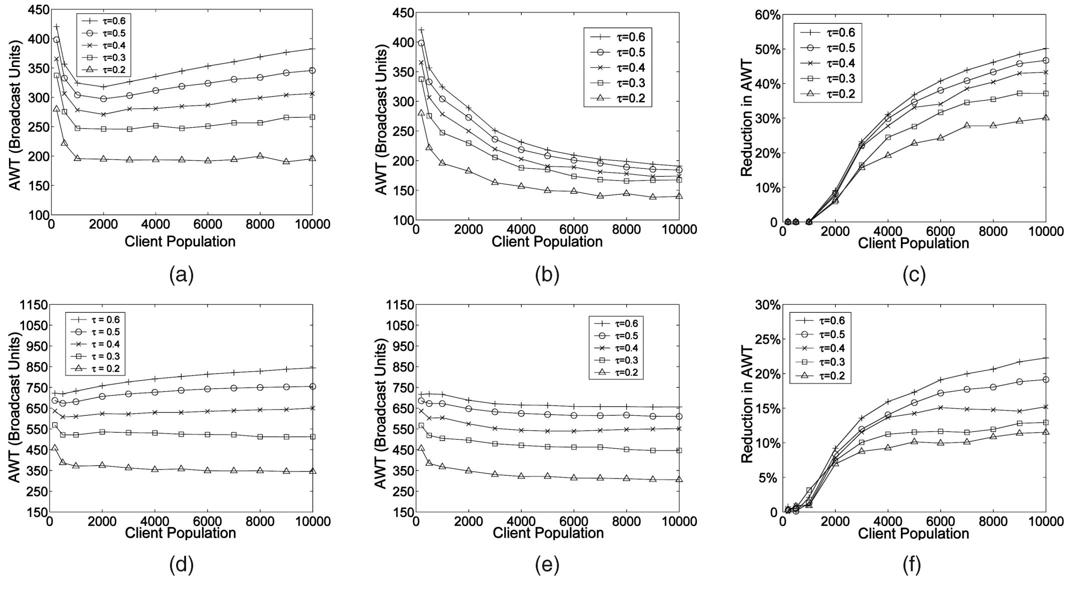


Fig. 9. Feedback methods compared: random sampling versus all clients responding. (a) All clients respond (Zipf). (b) Random sampling (Zipf). (c) Comparison (Zipf). (d) All clients respond (Uniform). (e) Random sampling (Uniform). (f) Comparison (Uniform).

uniform interest pattern is longer than under the Zipf interest pattern. Client requests are more clumped under Zipf so that the number of documents in the broadcast program becomes smaller.

Figs. 9a, 9b, and 9c show that the AWT initially decreases and increases after that. Under Zipf, a handful of documents account for most requests. Initially, with few clients, the set of requested documents, hence the broadcast size, remains relatively constant. In this phase, increasing the number of clients decreases the AWT. However, as the number of clients increases beyond a threshold, less popular documents will also tend to be requested, increasing the broadcast size. In this phase, we would not expect the AWT to continue dropping. As may be expected, these phases are less distinct for the uniform distribution than for Zipf.

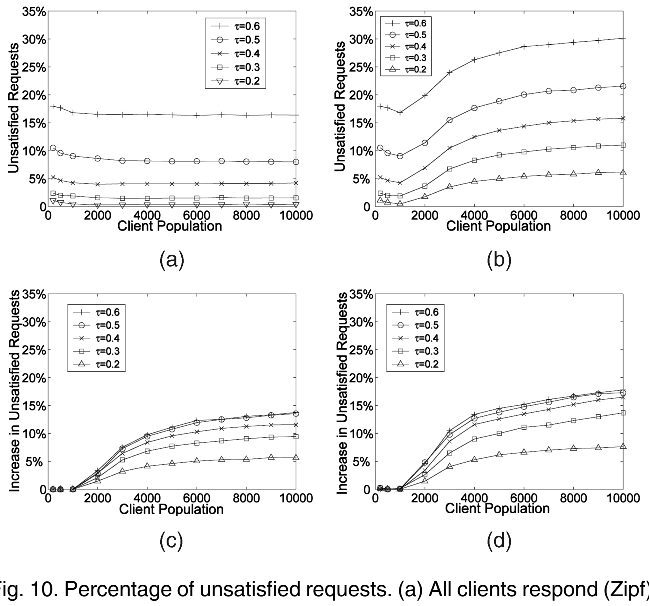


Fig. 10. Percentage of unsatisfied requests. (a) All clients respond (Zipf). (b) Random sampling (Zipf). (c) Comparison (Zipf). (d) Comparison (Uniform).

Fig. 10 shows the average percentage of unsatisfied requests in a broadcast cycle. Unsatisfied requests may arise for several reasons. First, no document in the database may match a client request when τ is relatively high (see Fig. 7). Second, the random sampling method in our model estimates the client interest 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 5.2. Fig. 10c shows an increase in the percentage of unsatisfied requests in our model. For $\tau = 0.6$ and 10,000 clients, about 13.8 percent of client requests may be unsatisfied with the broadcast.

Fig. 11 shows the superiority of our method in terms of the fraction of documents scheduled in the broadcast, which levels off beyond a client population of 2,000 for our approach. The results for uniform interest patterns show very similar trends. Our model includes fewer documents in the broadcast, so the broadcast frequency for each document can be high, reducing client waiting times. Our method is clearly more scalable.

Fig. 12 shows how adaptable our model is under shifting client interest patterns. In this experiment, the *ShiftFreq* is set to 10, and the shift *Offset* is set to 20. We observe relatively higher AWTs after the shifts in client interest pattern, but

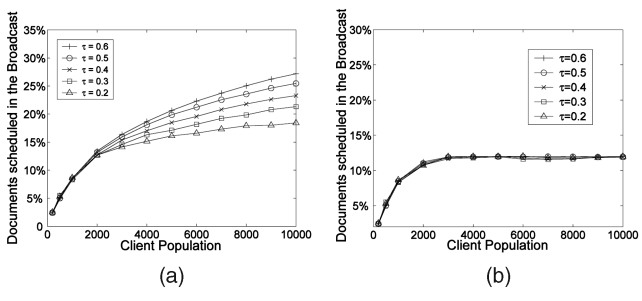


Fig. 11. Percentage of documents in server database scheduled in broadcast. (a) All clients respond. (b) Random sampling.

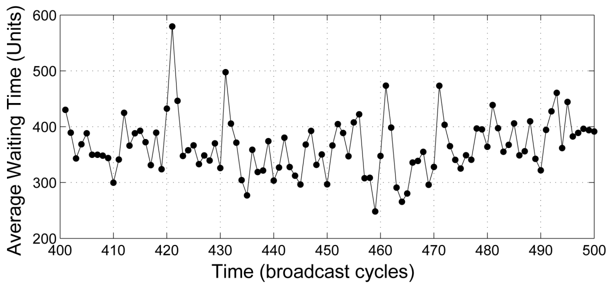


Fig. 12. Adaptability to shifts in client access pattern $(Shift = 10)$.

they drop back to normal very quickly. The shifts at cycles 421, 431, and 461 illustrate the point very well.

7.2.2 Comparison with Adaptive Broadcast Disk

Fig. 13 compares our model with the Adaptive Broadcast Disk (Adaptive BD) scheme in [17], which also uses bit-vector feedback. In Adaptive BD, only clients with explicit requests send feedback to the server. We use the same parameter settings as in the Adaptive BD model (see Table 2): these are entirely different from those in our previous experiments. Since all documents in the Adaptive BD model have a fixed size of 8,192 bytes, we ignore the actual document sizes in our database.

Clearly, our model is much more scalable than the Adaptive BD model. The AWTs in our model are also generally shorter than those for Adaptive BD. Adaptive BD allocates a fixed ratio of broadcast bandwidth for broadcasting on-demand requests so that the broadcast program can deviate from the client interest patterns. Our approach helps the server in detecting and exploiting client interest patterns much more precisely.

7.2.3 Selective Tune-In Performance

We evaluated our model with and without selective tune-in, broadcasting the same sequence of documents in both cases. That is, the index segments were interspersed within the sequence of documents broadcast in the model without indices. We evaluated both approaches described in Section 6.2. In the first method, all clients locate the requested documents by listening only to the index segments, switching to the doze mode if their query arrives in between two consecutive index segments. In the other method, a client listens to the broadcast until the first index segment arrives. If the document is not found by that time, it begins listening only to the index segments. We tested both the Zipf and Uniform client interest patterns for both models.

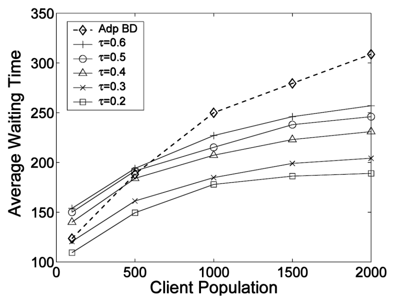


Fig. 13. Performance comparison with Adaptive BD.

TABLE 2
Parameter Settings

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

Fig. 14 compares the performance of the two models under the Zipf distribution. Figs. 14a and 14c show that the AWTs with the indexing scheme are moderately longer than those in the nonindex model. For example, the AWTs with indices in the first access method is at most 32.5 percent longer than the AWT without indices, with a 0.2 similarity threshold. The reason for the increased AWT is that the broadcast program is designed to minimize the AWT in the scheme without indices. However, as shown in Figs. 14b and 14d, the ATTs under the index scheme are improved significantly, since the clients must listen to the broadcast all the time when no indices are used. We see in Fig. 14b that the average tune-in times are improved by a factor of at least 2.5, a significant energy savings.

Fig. 15 shows that trends when the client interest pattern is uniform are similar to those under Zipf. Furthermore, the percentage of unsatisfied requests and the fraction database documents scheduled in the broadcast remained almost the same as they were without indices. Our indexing scheme increases waiting time modestly but reduces client tune-in times greatly, leading to considerable client energy savings.

8 Conclusions

We described an energy-efficient adaptive broadcasting model for asymmetric bandwidth environments and an

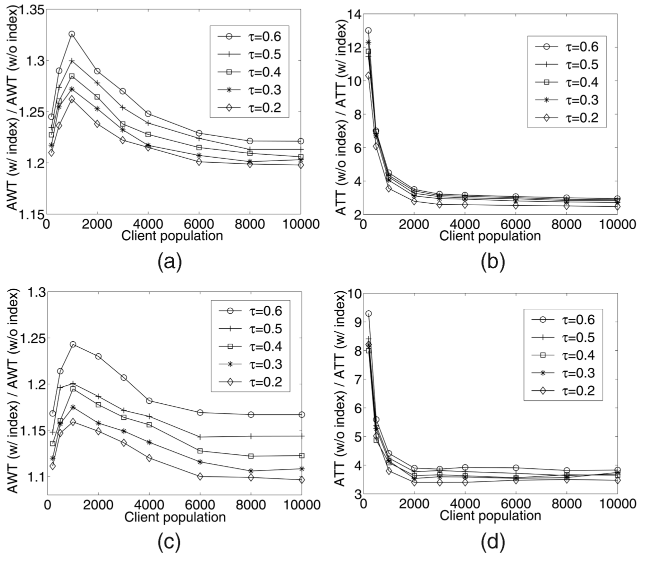


Fig. 14. Comparison under the Zipf client interest pattern. (a) Listen to the index only. (b) Listen to the index only. (c) Listen to the broadcast and index. (d) Listen to the broadcast and index.

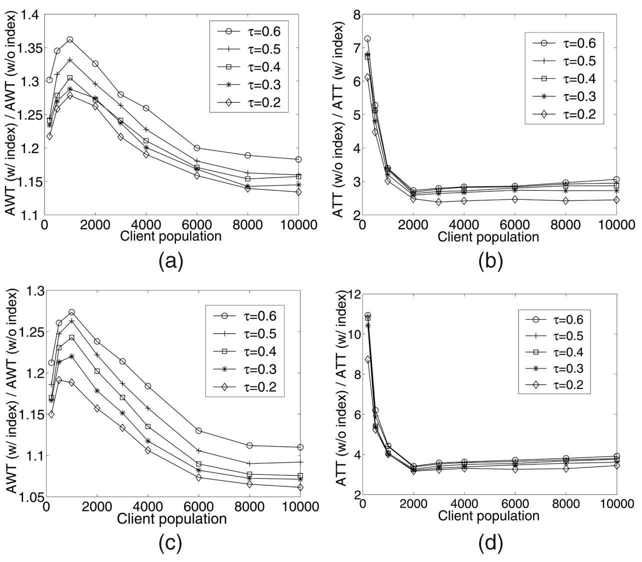


Fig. 15. Comparison under the uniform client interest pattern. (a) Listen to the index only. (b) Listen to the index only. (c) Listen to the broadcast and index. (d) Listen to the broadcast and index.

approximate response mechanism for queries. We also developed a randomized client feedback mechanism and a theory for bounding the client feedback sample size. Servers can estimate client interest patterns quickly, effectively, and efficiently. We also proposed an objective function for optimizing our model and showed how we can create a near-optimal broadcast program conforming to this objective function. Finally, we integrated an indexing scheme based on approximate matching, in which clients selectively tune in to the broadcast, resulting in considerable energy savings. Most importantly, these mechanisms are seamlessly integrated into our system.

We have evaluated the performance of our model with real-world data sets by using an extensive and accurate simulation testbed. Even without indices, 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. Our model clearly outperforms Adaptive BD. Our results also show that with indexing, our model can achieve much higher energy savings, with some moderate waiting time overhead.

ACKNOWLEDGMENTS

This work was supported by a grant from Tata Consultancy Services, Inc. A preliminary version of the paper appeared in the *Proceedings of the Eighth International Database Engineering and Applications Symposium*.

REFERENCES

[1] W. Wang and C.V. Ravishankar, "Adaptive Data Broadcasting in Asymmetric Communication Environments," *Proc. Eighth IEEE Int'l Database Eng. and Applications Symp. (IDEAS '04)*, pp. 27-36, 2004.

[2] B. Xu, O. Wolfson, and S. Chamberlain, "Cost Based Data Dissemination in Broadcast Networks with Disconnection," *Proc. Eighth Int'l Conf. Database Theory (ICDT '01)*, pp. 114-128, 2001.

[3] B. Xu, O. Wolfson, S. Chamberlain, and N. Rishe, "Cost-Based Data Dissemination in Satellite Networks," *Mobile Networks and Applications*, vol. 7, no. 1, pp. 49-66, 2002.

[4] W.G. Yee, S.B. Navathe, E. Omiecinski, and C. Jermaine, "Efficient Data Allocation over Multiple Channels at Broadcast Servers," *IEEE Trans. Computers*, Oct. 2002.

[5] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communications Environments," *Proc. ACM SIGMOD*, 1995.

[6] S. Hameed and N.H. Vaidya, "Efficient Algorithms for Scheduling Data Broadcast," *ACM/Baltzer J. Wireless Networking.*, vol. 5, no. 3, pp. 183-193, 1999.

[7] C.-J. Su and L. Tassiulas, "Joint Broadcast Scheduling and User's Cache Management for Efficient Information Delivery," *Proc. ACM MobiCom '98*, pp. 33-42, 1998.

[8] T. Hara, "Cooperative Caching by Mobile Clients in Push-Based Information Systems," *Proc. 11th ACM Int'l Conf. Information and Knowledge Management (CIKM '02)*, pp. 186-193, 2002.

[9] G. Herman, G. Gopal, K.C. Lee, and A. Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems," *Proc. ACM SIGMOD '87*, pp. 97-103, June 1987.

[10] T.F. Bowen, G. Gopal, G. Herman, T. Hickey, K.C. Lee, W.H. Mansfield, J. Raitz, and A. Weinrib, "The Datacycle Architecture," *Comm. ACM*, vol. 35, no. 12, Dec. 1992.

[11] Los Angeles Times, http://www.latimes.com/services/newspaper/mediacenter/la-mediacenter-20% 02-06.htmlstory, 2007.

[12] CNN's Newswatch, http://www.cnn.com/services/newswatch, 2007.

[13] Battlefield of the Future, http://www.airpower.maxwell.af.mil/airchronicles/battle/bftoc.html, 2007.

[14] S. Acharya, M. Franklin, and S. Zdonik, "Dissemination-Based Data Delivery Using Broadcast Disks," *IEEE Personal Comm.*, vol. 2, no. 6, 1995.

[15] S. Acharya, M. Franklin, and S. Zdonik, "Balancing Push and Pull for Data Broadcast," *Proc. ACM SIGMOD*, 1997.

[16] K. Stathatos, N. Roussopoulos, and J.S. Baras, "Adaptive Data Broadcast in Hybrid Networks," *Proc. 23rd Int'l Conf. Very Large Data Bases (VLDB)*, 1997.

[17] Q. Hu, D.-L. Lee, and W.-C. Lee, "Dynamic Data Delivery in Wireless Communication Environments," *Proc. W3C Workshop Mobile Data Access '98*, pp. 213-224, Nov. 1998.

[18] J.-H. Hu, K. Yeung, G. Feng, and K. Leung, "A Novel Push-and-Pull Hybrid Data Broadcast Scheme for Wireless Information Networks," *Proc. IEEE Int'l Conf. Comm. (ICC '00)*, vol. 3, pp. 1778-1782, 2000.

[19] J. Oh, K.A. Hua, and K. Prabhakara, "A New Broadcasting Technique for an Adaptive Hybrid Data Delivery in Wireless Mobile Network Environment," *Proc. Ninth IEEE Int'l Performance, Computing, and Comm. Conf. (IPCCC '00)*, pp. 361-367, 2000.

[20] N.H. Vaidya and S. Hameed, "Data Broadcast in Asymmetric Environments," *Proc. First Int'l Workshop Satellite-Based Information Services (WOSBIS '96)*, pp. 38-52, 1996.

[21] N.H. Vaidya and S. Hameed, "Scheduling Data Broadcast in Asymmetric Communication Environments," *Wireless Networks*, vol. 5, pp. 171-182, 1999.

[22] C.-J. Su, L. Tassiulas, and V. Tsotras, "Broadcast Scheduling for Information Distribution," *Wireless Networks*, vol. 5, no. 2, pp. 137-147, 1999.

[23] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy, "Adaptive Push-Pull: Disseminating Dynamic Web Data," *Proc. 10th Int'l World Wide Web Conf. (WWW '01)*, May 2001.

[24] C.-L. Hu and M.-S. Chen, "Dynamic Data Broadcasting with Traffic Awareness," *Proc. 22nd Intl' Conf. Distributed Computing Systems (ICDCS '02)*, 2002.

[25] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. IEEE INFOCOM '99*, pp. 126-134, 1999.

[26] G. Salton, A. Wong, and C.S. Yang, "A Vector Space Model for Automatic Indexing," *Comm. ACM*, pp. 613-620, 1975.

[27] U. Cetintemel, M.J. Franklin, and C.L. Giles, "Self-Adaptive User Profiles for Large-Scale Data Delivery," *Proc. 16th IEEE Int'l Conf. Data Eng. (ICDE '00)*, pp. 622-633, Feb. 2000.

[28] K.-L. Wu, P.S. Yu, and M.-S. Chen, "Energy-Efficient Caching for Wireless Mobile Computing," *Proc. 12th IEEE Int'l Conf. Data Eng. (ICDE '96),* 1996.
[29] D. Aksoy and M. Franklin, "Scheduling for Large-Scale On-Demand Data Broadcasting," *Proc. IEEE INFOCOM '98,* vol. 2, pp. 651-659, 1998.
[30] D. Aksoy and M. Franklin, "RXW: A Scheduling Approach for Large-Scale On-Demand Data Broadcast," *IEEE/ACM Trans. Networking,* vol. 7, no. 6, pp. 846-860, 1999.
[31] S. Acharya and S. Muthukrishnan, "Scheduling On-Demand Broadcasts: New Metrics and Algorithms," *Proc. ACM MobiCom '98,* pp. 43-54, 1998.
[32] G. Salton and M.J. McGill, *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.
[33] *Information Retrieval: Data Structures and Algorithms,* W.B. Frakes and R. Baeza-Yates, eds. Prentice Hall, 1992.
[34] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, 1988.
[35] C. Buckley, G. Salton, J. Allan, and A. Singhal, "Automatic Query Expansion Using SMART," *Proc. Third Text Retrieval Conf. (TREC '94),* pp. 69-80, 1994.
[36] B. Vélez, R. Weiss, M.A. Sheldon, and D.K. Gifford, "Fast and Effective Query Refinement," *Proc. 20th ACM SIGIR,* pp. 6-15, 1997.
[37] M. Mitra, A. Singhal, and C. Buckley, "Improving Automatic Query Expansion," *Proc. 21st ACM SIGIR,* pp. 206-214, 1998.
[38] R. Motwani and P. Raghavan, *Randomized Algorithms.* Cambridge Univ. Press, 1995.
[39] H. Chernoff, "A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations," *Annals of Math. Statistics,* vol. 23, no. 4, pp. 493-507, 1952.
[40] M.H. Ammar and J.W. Wong, "The Design of Teletext Broadcast Cycles," *Performance Evaluation,* vol. 5, no. 4, pp. 235-242, 1985.
[41] D.P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods.* Athena Scientific, 1996.
[42] S. Hameed and N.H. Vaidya, "Log-Time Algorithms for Scheduling Single- and Multiple-Channel Data Broadcast," *Proc. ACM MobiCom '97,* pp. 90-99, Sept. 1997.
[43] T. Imielinski and B.R. Badrinath, "Mobile Wireless Computing: Challenges in Data Management," *Comm. ACM,* vol. 37, no. 10, pp. 18-28, 1994.
[44] Lucent, *IEEE 802.11 waveLAN PC Card User's Guide,* 2007.
[45] T. Simunic, H. Vikalo, P. Glynn, and G.D. Micheli, "Energy Efficient Design of Portable Wireless Systems," *Proc. Int'l Symp. Low-Power Electronics and Design (ISPLED '00),* 2000.
[46] T. Imielinski, S. Viswanathan, and B.R. Badrinath, "Energy Efficient Indexing on Air," *Proc. ACM SIGMOD '94,* pp. 25-36, May 1994.
[47] T. Imielinski, S. Viswanathan, and B.R. Badrinath, "Data on Air: Organization and Access," *IEEE Trans. Knowledge and Data Eng.,* vol. 9, no. 3, May/June 1997.
[48] W.-C. Lee and D.L. Lee, "Using Signature Techniques for Information Filtering in Wireless and Mobile Environments," *Distributed and Parallel Databases,* vol. 4, no. 3, pp. 205-227, 1996.
[49] Q. Hu, W.-C. Lee, and D.L. Lee, "A Hybrid Index Technique for Power Efficient Data Broadcast," *Distributed and Parallel Databases,* vol. 9, pp. 151-177, 2001.
[50] S. Lee, D.P. Carney, and S. Zdonik, "Index Hint for On-Demand Broadcasting," *Proc. 19th IEEE Int'l Conf. Data Eng. (ICDE '03),* pp. 726-728, Mar. 2003.
[51] J.-L. Huang and W.-C. Peng, "An Energy-Conserved On-Demand Data Broadcasting System," *Proc. ACM SIGMOD '05,* pp. 234-238, 2005.
[52] CSIM 19 Simulation Engine, http://www.mesquite.com/documentation/, 2007.
[53] D.D. Lewis, "Reuters-21578, Distribution 1.0," http://www.daviddlewis.com/resources/, 2007.
[54] M. Portor, "The Portor Stemming Algorithm," http://www.tartarus.org/~martin/PorterStemmer/, 2007.
[55] I.S. Dhillon, J. Fan, and Y. Guan, "Efficient Clustering of Very Large Document Collections," *Data Mining for Scientific and Eng. Applications,* Kluwer Academic Publishers, 2001.
[56] I.S. Duff, R.G. Grimes, and J.G. Lewis, "Sparse Matrix Test Problems," *ACM Trans. Math. Software,* vol. 15, no. 1, pp. 1-14, 1989.
[57] D. Knuth, *The Art of Computer Programming, Volume II: Eminumerical Algorithms.* Addison Wesley, 1981.
[58] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," *Proc. ACM SIGCOMM '99,* pp. 251-262, 1999.

Wei Wang received the bachelor's and master's degrees in computer science and engineering from Northeastern University, Shenyang, China. She is currently working toward the PhD degree in the Department of Computer Science and Engineering, University of California, Riverside. Her research interests include distributed systems and networks. Her PhD studies are focused on wireless and mobile communication environments.

Chinya V. Ravishankar received the bachelor's degree in chemical engineering from the Indian Institute of Technology, Bombay, and the MS and PhD degrees in computer science from the University of Wisconsin, Madison. From 1986 to 1999, he was with the faculty of the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. Since the Fall of 1999, he has been with the University of California, Riverside, where he is currently a professor of computer science and engineering and the associate dean in the Bourns College of Engineering. His research interests include software systems, databases, networking, and security. He is a senior member of the IEEE and a member of the ACM.