Scheduling for Large-Scale On-Demand Data Broadcasting

Demet Aksoy, Michael Franklin
Computer Science Department and UMIACS
University of Maryland, College Park MD
demet@cs.umd.edu, franklin@cs.umd.edu

Abstract— **Recent advances in telecommunications have enabled the deployment of broadcast-based wide-area information services that provide on-demand data access to very large client populations. In order to effectively utilize a broadcast medium for such a service, it is necessary to have efficient, on-line scheduling algorithms that can balance individual and overall performance, and can scale in terms of data set sizes, client populations, and broadcast bandwidth. In this study we introduce a parameterized algorithm that provides good performance across all of these criteria and can be tuned to emphasize either average or worst case waiting time. Unlike previous work on low overhead scheduling, the algorithm is not based on estimates of the access probabilities of items, but rather, it makes scheduling decisions based on the current queue state, allowing it to easily adapt to changes in the intensity and distribution of the workload. We examine the performance of the algorithm using a simulation model.**

I. Introduction

A. On-demand Data Broadcast

Broadcast-based information systems are becoming increasingly popular due to advances in telecommunications, interconnectivity and mobile computing. Compared to traditional unicast data transfer, broadcasting can be much more efficient for disseminating information to large numbers of clients in applications where there is a high degree of commonality among client interests. With unicast, a data item must be transmitted at least once for every client who requests it, resulting in scalability problems as the client population increases. The World Wide Web has provided numerous examples of such situations, such as election result servers, sporting event kiosks, stock market tickers, etc. The access delays associated with such sites on the WWW during periods of heavy use demonstrate the inefficiencies of unicast delivery for dissemination-oriented applications.

The advantage of broadcast for data dissemination is that each transmission of an item can satisfy the needs of potentially many clients. Several forms of data broadcasting have already been introduced commercially. Intel has been broadcasting data along with normal TV signals [Intel94]. Hughes Network Systems is using satellites for delivering Internet content [DirecPC] and plans to incorporate broadcast technology. Broadcasting using cable technology is being developed by Hybrid Networks Inc. [Hybrid] and others. There has also been tremendous improvement in the bandwidth that is available for data broadcast. For example, the Teledesic system is expected to provide bandwidth of 155.52Mbps up to 1.244Gbps [Teledesic].

This work has been partially supported by the NSF under grant IRI-9501353, by Rome Labs agreement number F30602-97-2-0241 under DARPA order number F078, and by research grants from Intel and NEC.

B. Our Focus

While broadcast technology continues to advance in terms of both ubiquity and bandwidth, improvements in interconnectivity are fueling explosive growth in the amount of data available on-line and the number of clients who wish to access that data. In this paper, we focus on scheduling algorithms for *dissemination-oriented applications* in which a large and possibly changing client population requests data items from an information source equipped with a data broadcasting capability. The challenge in developing scheduling algorithms for such on-demand data broadcast is to provide scalable performance that balances average and individual (i.e., worst case) responsiveness using the shared broadcast medium. Such algorithms must cope with large databases, large client populations with dynamically changing interests and composition, and with high broadcast bandwidth.

As indicated by the preceding discussion, a scheduling approach for large-scale, on-demand data broadcast must balance different requirements. The algorithms that have been developed to date (e.g., [Dyke86], [Wong88], [Vaidya96], [Su97]) have failed to meet one or more of these needs. Some approaches have used simple scheduling policies such as FCFS (First Come First Served), which provide average case performance that is significantly lower than what could be supported by the broadcast medium. More sophisticated approaches aimed at providing better performance have been based on assumptions that limit their applicability, such as assuming very small database sizes, static data access probabilities (thereby limiting the ability to adapt to changing client needs), and/or ignoring the overheads associated with making intelligent scheduling decisions.

A key element that has been missing from the previous work is a comprehensive set of metrics for on-demand data broadcast in large-scale data dissemination environments. In this paper, therefore, we first outline the performance criteria that must be addressed by such scheduling algorithms. These criteria include: average and worst case performance, scheduling overhead, and robustness in the presence of certain environmental changes. We show how existing algorithms fail to meet one or more of these criteria.

We then define a parameterized algorithm, called *RxW*, that performs well for all of these metrics and furthermore, can be tuned to focus on scheduling overhead, average waiting time, or worst case wait time according to the needs of a particular application. *RxW* is robust to changes in the client population and workload because it makes scheduling decisions based only on the current queue state, rather than depending on estimates of data item access probabilities.

The remainder of the paper is structured as follows. In Section II we give a brief description of the problem and define the important criteria for evaluating scheduling algorithms for large-scale broadcast. We then describe how previously proposed algorithms measure up to these criteria. In Section III we develop several variants of a new algorithm, called RxW, which has low overhead and provides good average and worst case performance. Section IV presents an evaluation of the algorithm in terms of its performance, scalability, and robustness to workload changes. Section V discusses related work. Finally, Section VI presents our conclusions.

II. Background

A. Environment

In this section we present a simple satellite-based broadcast scenario to motivate the scheduling problem we are addressing. In this scenario (depicted in Figure 1) clients use two independent networks for communicating with the server: a terrestrial network for sending requests to the server, and a "listen only" satellite downlink to receive data from the server, similar to Hughes Network System's DirecPC architecture [DirecPC] and other satellite data services. When a client needs a data item (e.g., a web page or database object) that it cannot find locally, it sends a request for the item to the server. Client requests are queued up (if necessary) at the server upon arrival. The server repeatedly chooses an item from among these requests, broadcasts it over the satellite link, and removes the associated request(s) from the queue. Clients monitor the broadcast and receive the item(s) that they are waiting for.

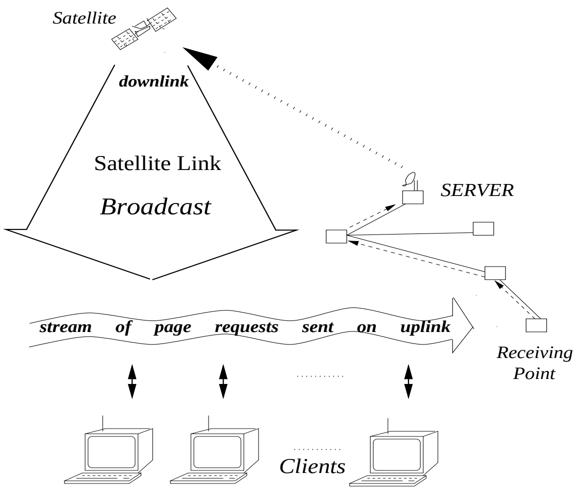


Fig. 1. Example Data Broadcasting Scenario

The focus of this paper is on the *scheduling* algorithm used by the server to choose the item to broadcast among those that have been requested. Because a single broadcast of an item satisfies *all* of the outstanding requests for the item, a good scheduling algorithm has the potential to greatly improve the effectiveness of the broadcast.

Similar to the previous work on broadcast scheduling we make the following assumptions about the environment. First, we assume that data items are fixed-length (e.g., database pages) so that the broadcast bandwidth can be divided into equal length, item-sized "slots". In the remainder of the paper, we refer to such fixed-sized items as "pages" and we refer to the length (in time) of a broadcast slot as a *broadcast tick* and use such ticks as the unit of time measure.[1] Second, we assume that clients continuously monitor the broadcast after they make a request and we do not consider the effects of transmission errors, so that all clients that are waiting for an item receive that item when it is broadcast by the server. Finally, we ignore the delay for sending requests via the client-to-server uplink, which we expect to be small compared to the latency of obtaining broadcast items from a moderately or heavily loaded server.

B. Performance Issues

Given the application environment described so far, we can now state our criteria for evaluating broadcast scheduling algorithms for large-scale data dissemination.

B.1 Responsiveness

The success of a scheduling algorithm is determined by its ability to get requested data to the clients quickly. In this regard, the first important metric, *average waiting time* is the amount of time on average, from the instant that a client request arrives at the server, to the time that the requested item is broadcast. Second, *worst case waiting time* is the maximum amount of time that any client request will have to wait in the service queue to be satisfied.

B.2 Scheduling Overhead

Because of the requirement for scalability, a key aspect of this study is the consideration of scheduling overhead at the server. Overhead is examined in two categories:

1. *Request Processing* - When a new request arrives at the server, the server must quickly decide whether or not to place an entry in the request queue for the requested item and/or update and possibly restructure the queue contents. The speed of such processing limits the rate at which requests can be processed by the server, effectively placing a limit on the scalability in terms of request arrival rate (e.g., number of clients supported).
2. *Scheduling Decisions* - On every broadcast tick the server must choose a page to broadcast. If the decision overhead is excessive the server may not be able to support the increased broadcast bandwidth or larger database sizes.

The two types of scheduling overhead are related: for example, doing extra work when requests arrive (e.g., keeping sorted lists of page access probabilities) can reduce the cost of making scheduling decisions at the expense of an increased cost for processing requests. The proper trade-off between these costs is dictated by the types of scalability that are important for a particular environment.

[1] The fixed length assumption simplifies the algorithm descriptions and analysis. Recent work in broadcast scheduling has shown how to extend scheduling algorithms to incorporate variable-length items [Vaidya96], [Su97].

B.3 Robustness

In order to achieve the goals of responsiveness and scalability, a scheduling algorithm will typically employ approximations and/or heuristics. Such heuristics must not be based on static information that will cause the algorithm to perform poorly if the workload or the environment changes.

C. *Previous Algorithms*

As stated in the Introduction, several algorithms for on-demand broadcast scheduling have been proposed previously. In this section, we describe existing algorithms and discuss their limitations with respect to the criteria that were outlined in the preceding section. Dykeman et al. [Dyke86] studied on-line scheduling algorithms, and were the first to point out that traditional FCFS scheduling would provide poor average wait time for a broadcast environment when the access distribution for data items was non-uniform. They proposed several algorithms aimed at providing improved performance. The algorithms studied in [Dyke86] (and later described in [Wong88]) are the following:

- **First Come First Served (FCFS)**: broadcasts the pages in the order they are requested.
- **Most Requests First (MRF)**: broadcasts the page with the maximum number of pending requests.
- **Most Requests First Lowest (MRFL)**: is essentially same as MRF, but breaks ties in favor of the page with the lowest access probability.
- **Longest Wait First (LWF)**: selects the page that has the largest total waiting time, i.e., the sum of the time that all pending requests for the item have been waiting.

Figure 2 plots the average waiting time (in broadcast ticks) for a workload with a database of 10000 pages. Client requests for pages are generated using a Zipf distribution of maximum skewness (parameter θ=1). The results were generated using the simulation environment and default parameters that we will describe in Section IV. As in [Dyke86], the overheads associated with running the scheduling algorithm at the server are not modeled here.

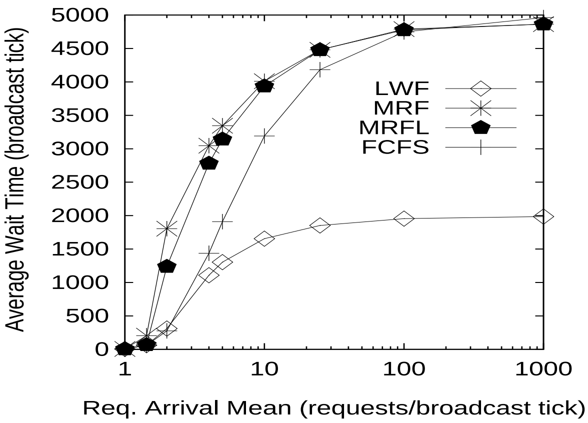


Fig. 2. Average Wait Time for Algorithms of Dykeman et al.

As can be seen in the figure, the best performance overall in this case is provided by LWF. As would be expected, the average wait time increases for all algorithms as more requests are introduced. Perhaps less predictably, however, the average response time eventually levels off and becomes insensitive to additional load. At this point, the remaining algorithms are approximately 2.5 times slower than LWF. Unfortunately, LWF is not a practical algorithm for a large system, as at each broadcast tick, it recalculates total accumulated wait time for *every* page with pending requests in order to decide which page to broadcast. For a high-bandwidth system with a large database, such a scheduling algorithm would likely become a bottleneck.[2] MRF and MRFL algorithms were introduced as lower-overhead alternatives.

The results of Figure 2 agree with those of [Dyke86], [Wong88] except for two key points. First, the earlier work did not investigate the performance of the algorithms under very high loads, so it did not identify the flattening of the performance curves for all of the algorithms under high load. Second, in the earlier study MRFL was seen to provide a performance between that of FCFS and LWF. Thus, MRFL was proposed as a lower-overhead replacement for LWF. In contrast, our results show that MRFL has poor performance relative to LWF and thus, is not a reasonable replacement. The differences in the conclusions stem from the fact that the earlier study was performed using a very small database (100 items, compared to 10000 in Figure 2). As the size of the database increases, the probability of having a tie for the largest number of requests diminishes. Without ties, MRFL degenerates to MRF, and has relatively poor performance.

The poor performance of MRFL for larger systems has also recently been shown by Su and Tassiulas [Su97]. In that paper, they propose an alternative algorithm, called PIP-0.5 (Priority Index Policy), that performs as well as LWF in average wait time. Unlike LWF, PIP-0.5 can be implemented with an O(1) complexity for choosing the next page to broadcast. PIP-0.5 falls short of our performance criteria, because it is based on estimates of the probability of access for each item. As a result, its usefulness is limited to fairly stable environments where those probabilities do not often change significantly. Furthermore, the history mechanism that must be employed to obtain such probability estimates can result in additional overhead, particularly for very large data sets. Algorithms based on access probabilities and broadcast histories have also been proposed by Vaidya and Hameed [Vaidya96]. These algorithms have similar performance to the PIP-0.5 algorithm, and also share that algorithm's limitations in terms of robustness to changing workloads.

III. RxW: A Parameterized Algorithm

We now describe a new broadcast scheduling algorithm, called RxW, which is a practical, low-overhead, scalable approach that provides excellent performance across a range of scenarios.

[2] In our implementation of LWF, we found that using one processor of a DEC Alpha 2100 4/275 server and assuming a broadcast bandwidth of 155.52 Mbps, the LWF algorithm became a bottleneck with a database size of 5543 8KByte pages.

A. Intuition

The results shown in Figure 2 demonstrated that MRF and FCFS have poor average case performance compared to the higher-overhead LWF algorithm. Probing more deeply into their performance in this case leads to a very important observation. Figures 3(a) and 3(b) show the performance of the three algorithms for the 10% most popular pages in the database (i.e., the "hot" pages), and the remaining 90% of the pages (i.e., the "cold" pages) respectively.[3]

As can be seen in the figures, MRF provides the lowest waiting time for hot pages, but its performance for cold pages is by far the worst of the three algorithms. In contrast, FCFS provides similar performance for both classes of pages leading it to have the worst performance for hot pages and the best for cold pages of the three. MRF chooses the page with the highest number of outstanding requests, so that requests for infrequently accessed pages must wait until sufficient requests have arrived. Since MRF is not a starvation-free algorithm; it is quite possible that a request for a very cold page is never satisfied. In contrast, FCFS is a fair algorithm in which the maximum time a request must wait is the same for all pages. This behavior causes it to spend more bandwidth on requests for cold pages. The fact that both algorithms favor one class of pages over the other results in their both having poor performance on average. In contrast, LWF provides good performance for both types of pages, resulting in better average performance overall. Based on these observations, we set out to combine the two low-overhead approaches (MRF and FCFS) in a way that would balance their strengths and weaknesses and provide a more even-handed treatment of hot and cold pages.

B. Scheduling with RxW

B.1 The Exhaustive Algorithm

We have developed a new scheduling algorithm called *RxW* (Requests times Wait), which provides good performance by combining the benefits of MRF and FCFS in a way that ensures scalability by having low overhead. *RxW* broadcasts a page either because it is very popular or because it has at least one long-outstanding request.

RxW maintains a service queue structure with a single entry for each page that has outstanding requests. Entries contain a page identifier (**PID**), the count of the number of outstanding requests (**REQcnt**), and a time-stamp of the earliest unsatisfied request for the item (**1stARV**). This structure is hashed on PID.

The exhaustive algorithm works as follows: *When a request arrives* at the server, the server performs a hash look up to find the entry of the requested page. If it finds an entry, then it simply increments the **REQcnt** for that entry. If no entry is found, then a new one is created with **REQcnt** initialized to 1, and **1stARV** initialized to the current time. *At each broadcast tick*, the server chooses to broadcast the page with the largest value of $(R * W)$ with $R =$ **REQcnt** and $W = clock -$ **1stARV**, where *clock* is the current time in broadcast ticks. The entry for this page is then removed from the structure. The exhaustive algorithm finds the page with the maximum value by simply performing a linear scan of *all* the entries. Note that the queue size is limited by N, the number of pages in the database. Thus, the exhaustive *RxW* algorithm is similar in overhead to other proposed O(N) algorithms, such as [Vaidya96]. Note however, that *RxW* makes decisions based only on the current state of outstanding requests, and does not depend on estimates of page access probabilities. A detailed analytical study of the average waiting time and the limiting behavior of *RxW* is provided in [Aksoy97].

[3] Recall that the popularity of the pages for this case were generated according to the Zipf distribution.

B.2 The Pruning Algorithm

As with other O(N) scheduling algorithms, the overhead of *RxW* scheduling can be reduced by performing more work during request processing in order to keep the request information better organized. In order to avoid searching the entire list of pages with outstanding requests, we thread two sorted lists through the request queue structure: The **Wait** list is simply a FCFS queue based on **1stARV** in ascending order; the second list is the **Requests** list, which is kept sorted in descending order by **REQcnt**.

The **Wait** queue is maintained by simply appending a request entry to it when a request arrives for a page with no outstanding requests and removing a page's entry when the page is broadcast. The **Requests** list is maintained each time a request is received. This maintenance involves moving the affected entry to the proper place in the sorted list. We introduce an additional structure to speed up the maintenance process and to guarantee scalability in request arrival rates. This structure, called the "REQcnt index" contains pointers to each cluster of **REQcnt** value, i.e. all pages that have the same number of pending requests. Values are added to and removed from this index as the request structure evolves. Using this index, request processing is an O(1) operation (note that for any one request, an entry moves exactly one cluster up in the **Requests** list).

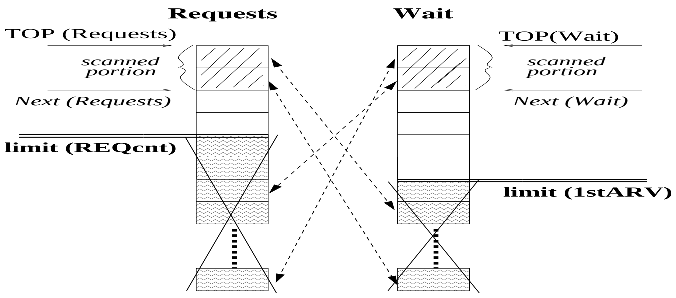
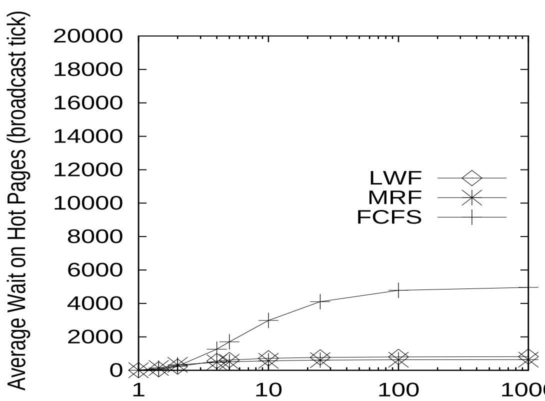
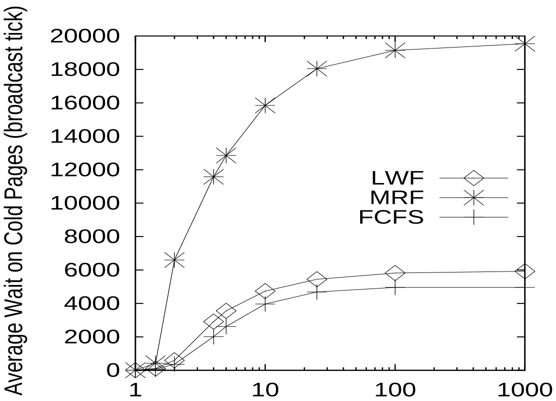


Fig. 4. Pruning the Search Space

The two sorted lists, (**Requests** and **Wait**), are used to prune parts of each list where there is no possibility of containing the entry with the highest *RxW* value. This pruning technique is depicted in Figure 4. Note that nodes connected by a dashed line belong to a single service queue entry. On the left is the **Requests** list, ordered by descending **REQcnt** and on the right is the **Wait** queue, ordered by ascending



(a) Hot Pages



(b) Cold Pages

Fig. 3. Average Waiting Time for Hot and Cold Page Requests

1stARV. When an entry is examined from the **Requests** queue, it is known that all entries not yet scanned have a **REQcnt** field that is less than or equal to that of the entry. Thus, if the recently examined entry has values $REQcnt'$ and $1stARV'$, the only way for a remaining entry to beat the current maximum RxW value seen so far (MAX), is if

$$clock - 1stARV \geq \frac{MAX}{REQcnt'}$$

where $clock$ is the current time. Thus, the entries that must be searched on the **Wait** queue are bounded by:

$$limit(1stARV) = min(1stARV', clock - \frac{MAX}{REQcnt'})$$

The same kind of pruning can be applied on the **Requests** list by scanning entries on the **Wait** queue. The pruning algorithm starts from the top of the **Request** list (thereby truncating the **Wait** queue) and then examining the top of the **Wait** queue (thereby truncating the **Requests** list) and alternating until the search reaches the bottom of one of the (truncated) lists. Note that the stopping condition is checked merely by comparing the limit values, rather than actually maintaining a pointer to the exact boundary on the list.

This mechanism prunes the search space while still guaranteeing that the search will return the page with the maximum RxW value. Thus, in the performance study that follows, we ignore the exhaustive algorithm and use the term "RxW algorithm" to refer to this pruning variant.

B.3 Approximation for Speed Up

Scheduling based on the highest RxW value makes efficient use of broadcast bandwidth. Such efficiency, however, comes at a price in terms of the overhead that is incurred at each scheduling decision. As stated in Section II-B, such overhead can ultimately limit scalability in terms of supportable bandwidth or database size. The pruning technique for RxW aims to reduce this overhead. As will be shown in Section IV-C, this technique is indeed effective — for example, in the main workload of our experiments this pruning resulted in a 72% savings in terms of the number of entries searched in order to find the maximum RxW-valued page. While such a substantial savings is helpful, it is probably not sufficient to keep the scheduling overhead from ultimately becoming a limiting factor as the system is scaled to the huge applications that will be enabled by the national and global broadcasting systems currently being deployed.

Based on insight gained from early experiments with the RxW algorithm, we realized that the scheduling overhead can be reduced dramatically by backing off from the requirement of maximality for the RxW value when making scheduling decisions. We have developed an *approximate*, parameterized variant of RxW that allows the search space to be reduced further, at the possible expense of making less efficient use of the broadcast bandwidth. By varying a single parameter, the algorithm can be tuned from having the same behavior of the RxW algorithm described so far, to a *constant time* approach that provides maximal scalability.

The parameterized version of RxW is based on two insights about RxW scheduling. First, we found that with highly skewed access patterns (as would be expected in many dissemination-oriented applications), the page with the maximum RxW value is typically found very near the top of at least one of the two lists (Requests or Wait). As a result, even the pruning-based RxW algorithm can spend substantial resources examining entries after it has already encountered the maximum-valued entry. The second insight is that given a static workload (i.e., in terms of request arrival rate and access probability distribution) the average RxW value of the page chosen for broadcast typically converges to some value. This latter insight is exploited to create a *self-adapting* approximation algorithm based on the RxW value of the most recently broadcast page. We take care, however, to ensure that the approximation works well even in the presence of a changing workload.

The approximation algorithm requires a single parameter called *accuracy*, which can be set to any value, 0 or greater.[4]

[4] Typically the accuracy parameter will be set to a value between 0 and 1. Larger values can be used, however. In the limit, setting accuracy

Scheduling works as follows: First, the algorithm maintains a self-adapting *threshold*, which is updated on every broadcast tick by averaging the current threshold value with the RxW value of the page that was chosen to be broadcast on that tick. To find the next page to broadcast the request structure is searched as in the regular (pruning) RxW algorithm, but rather than searching for the page with the *maximal* RxW value, the algorithm chooses the *first* page it encounters whose RxW value that is greater than or equal to *accuracy* $\times$ *threshold*. If no such page is found, then the algorithm acts like the regular RxW algorithm and returns the page with the maximum RxW value.

The setting of the *accuracy* parameter determines the performance tradeoffs between average waiting time, worst case waiting time, and scheduling overhead. The smaller the value of the parameter, the fewer entries are likely to be scanned. At an extreme value of 0, the algorithm simply compares the top entry from both the **Requests** list and the **Wait** queue and chooses the one with the highest RxW value. In this case, the complexity of making a scheduling decision is reduced to O(1), ensuring that broadcast scheduling will not become a bottleneck regardless of the broadcast bandwidth, database size, or workload intensity. In the following section, we examine the performance tradeoffs of several settings for the *accuracy* parameter. We refer to the approximation-based RxW algorithms as $RxW.\alpha$, where α equals the value of the *accuracy* parameter as a percentage (e.g., an α setting of 0.80 is called $RxW.80$).

IV. Experimental Results

A. Simulation Environment

Our experiments were performed using a simple simulation model of the system using CSIM [Schw86]. As with previous studies, the model is intended to capture only the quality of the schedule produced by the given scheduling algorithms. As such, it does not include the overheads of scheduling and request processing at the server. These costs have been described in the previous sections and are addressed in the scalability portion of the experiments (Section IV-C). Also in keeping with earlier studies, we do not model the costs of using the back-channel for sending requests from the clients to the server as these costs will be the same for all of the scheduling algorithms.[5] The broadcast channel is modeled as a server with a fixed rate of broadcast. We do not specify an absolute value for this rate, but rather, use *broadcast ticks* as our measure of time. This approach emphasizes that the results are not limited to any particular bandwidth and/or data item size, but rather, that they describe fundamental tradeoffs among the algorithms.

In the model, the client population is represented by a single request stream. The client population model generates non-blocking requests with exponential inter-arrival times with mean λ. We use an open system model since our work is aimed at supporting extremely large, highly dynamic client populations, and such client populations cannot be realistically modeled with a closed simulation system. The request pattern is shaped with a Zipf distribution [Knuth81]. This is a frequently used distribution for non-uniform data access. It produces access patterns that become increasingly skewed as its θ parameter increases from 0 (uniform access probability) to 1 (highly skewed). The *offset* and *freq* parameters are used to simulate interest shifts of the client population and the frequency of such shifts. The parameters and their settings are summarized in Table I.

B. Responsiveness

In the first experiment, we examine the responsiveness of several variants of the RxW algorithm, and compare them to the LWF and FCFS algorithms of Dykeman et al. As stated previously, the results we present here are measures of the quality of the scheduling choices made by the various algorithms, and do not take into account the overhead of scheduling and request processing. Under such assumptions, LWF (and similar algorithms such as PIP-0.5) have provided the best average case performance in previous studies. We report results for four variants of RxW: The pruning algorithm and the approximate algorithm with α values of 0.90, 0.80, and 0 (referred to as $RxW.90$, $RxW.80, and RxW.0$, respectively). Recall that $RxW.0$ examines only the top entry of each the two sorted lists of requests, and thus, makes scheduling decisions in constant time. The four variants of RxW allow us to investigate the tradeoffs between the exact and approximate approach, among the various α settings.

In Figure 5(a), we plot the *average* waiting time for each scheduling algorithm, as the mean request arrival rate is varied from 1 per tick to 1000 per tick along the x-axis (shown with a log scale). All algorithms exhibit similar performance here, with the average wait time increasing but ultimately leveling off as the request arrival rate is increased. This leveling off is a characteristic of broadcast data delivery to clients with shared interests and differs dramatically from what would be expected in a unicast environment.

Comparing the six algorithms, it can be seen that LWF and the RxW algorithm provide the best average performance (RxW even does slightly better for loads between 5 and 50 requests/tick). The good performance of RxW in this case demonstrates that the scheduling decision metric used by RxW is a reasonable substitute for that of LWF (the total waiting time), and even for perfect knowledge of access probabilities (as used by PIP-0.5, which also has similar performance to LWF). By far the slowest average performance in this case is provided by FCFS. As described in Section III-A, for a sufficiently loaded system FCFS allocates the same bandwidth to all accessed pages, regardless of their popularity, resulting in poor utilization of the broadcast.

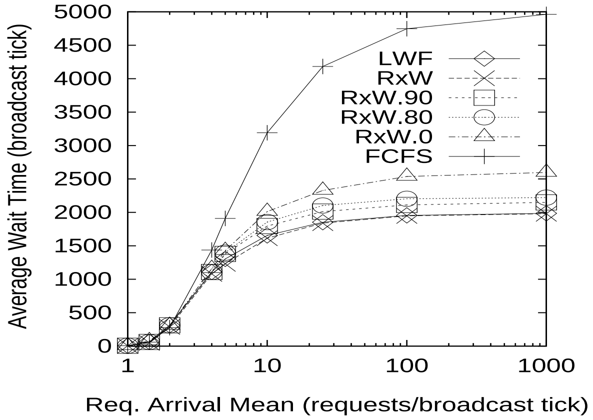
The results for the approximate RxW algorithm show that as the α parameter, which sets the accuracy for the broadcast schedule, is decreased the average wait time increases. For all three of the values shown, however, the performance

to ∞ results in the approximate algorithm behaving identically to the regular pruning-based RxW algorithm.

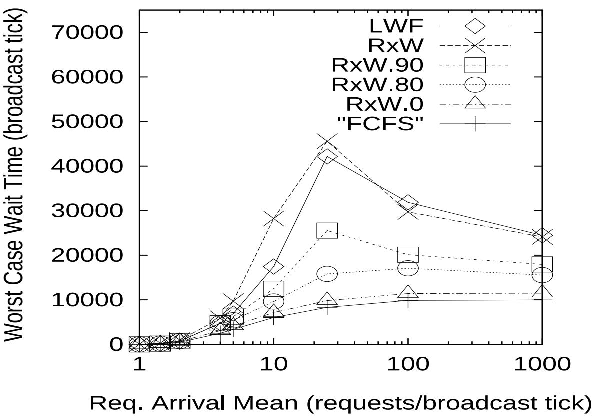
[5] The cost of back-channel requests becomes more important when trading off between server push and client pull over the broadcast as in [Acha97].

Symbol	Description	Default	Range	Unit
dbSize	Database Size	**10000**	[100-50000]	pages
λ	Mean Req Arv (exp)	**100**	[1-1000]	requests/tick
θ	Request Skew (zipf)	**1.0**	[0.0-1.0]	-
α	Approximation Accuracy	-	0, 0.80, 0.90	-
offset	Shift in Interest	**0**	[0-5000]	number of pages
freq	Interest Shift Freq	**n/a**	[1-100000]	/ 100000 broadcast ticks

TABLE I
SIMULATION MODEL PARAMETERS



(a) Average



(b) Worst Case

Fig. 5. Responsiveness Measures

is significantly better than that of FCFS. The approximate algorithm with an α value of 0.90 remains less than 10% slower than the maximal algorithm. Even the constant-time $RxW.0$ pays less than a 33% penalty compared to the maximal algorithm in the most extreme case here.

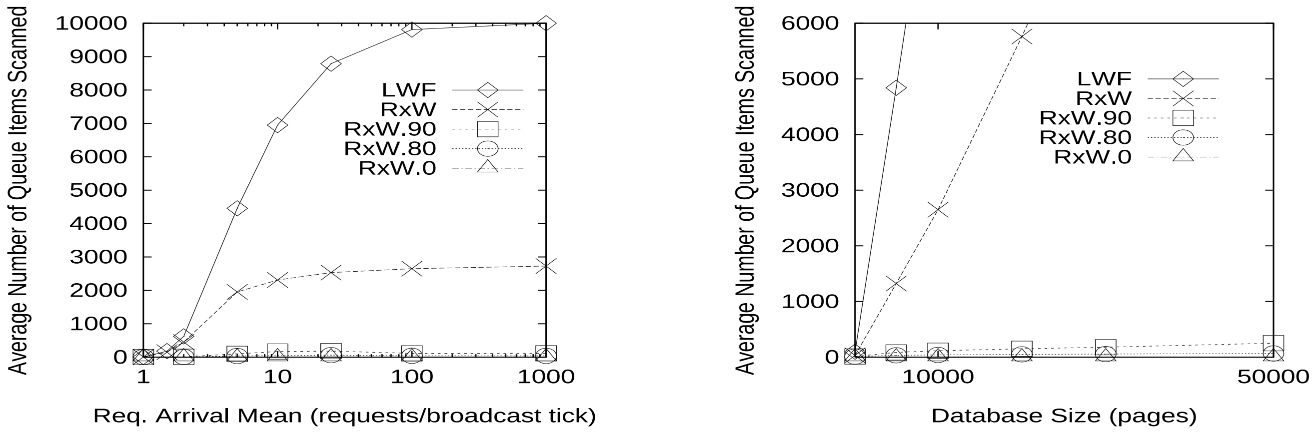
Figure 5(b) shows the *worst case* waiting time measured for the same experiment as Figure 5(a). That is, we plot the longest measured wait for *any* request that occurs during the simulation run. Note that the simulation was run one million broadcast ticks, so each page was broadcast at least several times. Although worst case performance has not been addressed by previous studies, it is an important metric for many applications. As can be seen in the figure, the ordering of the algorithms for worst case behavior is inverted compared to the average case. FCFS has the shortest worst case waiting time. In fact, once a page has been requested, it is guaranteed to be scheduled for broadcast before any other page is broadcast twice. Thus, its worst case behavior is bounded by the number of pages (10,000 in this case). In contrast, the LWF and RxW algorithms make no such guarantees — popular pages may be broadcast multiple times while requests for a less popular page are waiting.

For the approximate RxW algorithm, α serves as a knob for adjusting the worst case waiting time in the opposite direction of how it works for the average case. As α is decreased, the FCFS queue begins to play a larger role in the scheduling process, and thus, the behavior begins to look more like FCFS. As a result, for $\alpha = 0$, the worst-case behavior is within 15% of that of FCFS. Comparing Figures 5(a) and (b), it is apparent that the α parameter provides a flexible mechanism for trading-off worst case and average case waiting times for a particular application environment, and that it can also be set to balance both concerns reasonably well (e.g., $\alpha = 0.80$ in this case). In the next section we show that α can also be used to adjust the overhead of the scheduling decision process in order improve scalability.

C. Scheduling Overhead

As described previously, a critical aspect of scheduling algorithms for large-scale data broadcasting is scalability. The previous section focused on the performance of the algorithms in an ideal setting where there was no overhead for making scheduling decisions or processing requests. In practice, however, such concerns can limit the ability for on-demand systems to support large applications. As described in Sections II-C and III, all of the algorithms are fairly efficient in terms of request processing. They differ significantly, however, in terms of scheduling overhead. In this section we examine the question of scheduling overhead in more detail.

Figures 6(a) and (b) show the average number of request queue entries searched each time a scheduling decision is to be made (i.e., on each broadcast tick), as the request arrival rate and the database size are increased, respectively. Figure 6(a) corresponds to the same settings as the previous two graphs; that is, the database size is fixed at 10,000 pages and the request arrival rate is varied from 1 to 1000 requests/tick. As can be seen in the figure, LWF is by far the most expensive of the algorithms shown for making scheduling decisions, followed by the maximal RxW. Also, it can be seen that the



(a) Varying Request Rate (dbSize=10,000 pages)

(b) Varying Database Size (100 requests/tick)

Fig. 6. Scheduling Overhead Measures

approximate version of RxW provides tremendous savings.[6]

LWF is an exhaustive algorithm, and under high loads, there is at least one pending request for each data page. Thus, the scheduling cost of LWF is proportional to the number of distinct pages that are accessed by the client population, namely 10,000 pages in this case. For a fast broadcast bandwidth and/or large database size, the scheduling overhead of LWF could easily become a bottleneck. The maximal RxW algorithm, on the other hand, examines significantly fewer queue entries for each scheduling decision. In this case, it examines 2729 entries on average at a load of 1000 requests/tick; a savings of about 72.7%. It is important to note that as was shown in Figures 5(a) and (b), these savings in search complexity come at no cost in broadcast efficiency.

The savings provided by RxW's pruning algorithm, however, are dwarfed by the tremendous savings provided by the approximate version of the algorithm. In figure 5(a) at a load of 1000 requests/tick, RxW.90 and RxW.80 examine 116 and 39 entries respectively, for savings of more than 98.8% and 99.6% respectively. With α set to 0, the approximate algorithm scans only two entries providing maximum scalability in terms of search overhead.

Figure 6(b) shows even more striking results for the same algorithms, when the request rate is fixed at 100 requests/tick and the database is scaled from 1 to 50,000 pages. The overhead of LWF grows linearly with the database size, approaching the limit of one entry per page. The overhead of maximal RxW also grows linearly, but at a much slower rate. Finally, similarly to part (a) of the figure, the overhead of the approximate algorithms grows much more slowly, with RxW.0 remaining constant. The practical impact of these results is that the approximate RxW algorithm provides tremendous scalability in terms of request arrival rate and database size. In addition, although not shown here directly, these results indicate that RxW allows a broadcast system to scale in terms of the supportable broadcast bandwidth. Faster broadcast means shorter ticks, and thus, less time to make scheduling decisions. RxW is clearly capable of making fast scheduling decisions across a large range of system sizes and workload intensities.

[6] Note that FCFS is not shown on these graphs. It is a O(1) scheduling algorithm and so is insensitive to the parameters varied here.

D. *Robustness*

RxW and its approximations do not depend on any long term measurements or estimates of data access probability distributions, which enables them to easily adapt to changes in workload. The approximations, however, do use a threshold value that is dependent on previous performance. To test the robustness of the RxW variants, we performed a detailed sensitivity analysis. Due to space constraints, we briefly summarize the results of that analysis here. The results are reported in more detail in [Aksoy97].

In one set of experiments, the skewness of the access pattern (i.e., θ) was varied between 1 (default) and 0 (a uniform distribution). As the skew is reduced, all of the algorithms converge to the same average waiting time (at $\theta = 0$). The relative ordering of the algorithms studied remained constant. A second set of experiments kept θ at 1 but varied the focus of interest (i.e., the most popular items) to different parts of the database. The results showed that the approximate variants of RxW were slightly less robust than LWF for infrequent shifts of interest but were significantly more robust than LWF as the frequency of interest shifts was increased. Finally, a third set of experiments introduced a sudden 20-fold spike or a 20-fold decrease in the arrival rate of client requests. The results showed that the number of request queue entries scanned by RxW and its approximations still remained far below that of LWF.

V. Related Work

In this paper we have presented a new on-demand scheduling algorithm for large-scale data broadcast. The directly relevant previous work on scheduling algorithms [Dyke86], [Wong88], [Vaidya96], [Su97] has been addressed in detail in Section II. In addition to this directly related work, there has been much recent interest in other areas of data broadcasting. A taxonomy of data delivery mechanisms (including various forms of broadcast) along with a framework for describing dissemination-based systems is provided in [Fran97]. Some recent applications of dissemination-based systems include information dissemination on the Internet [Yan96], [Best96], Advanced Traveler Information Systems [Shekhar96] and dissemination using satellite networks [Dao96].

The Datacycle Project [Herm87], [Bowen92] at Bellcore investigated the use of a repetitive broadcast medium for database storage and query processing. An other early effort in information broadcasting, the Boston Community Information System (BCIS) is described in [Giff90]. BCIS broadcast news articles and information over an FM channel to clients with personal computers specially equipped with radio receivers. Recently, scheduling techniques from the real-time community have been applied to data broadcast by Baruah and Bestavros [Baru96]. The Broadcast Disks project [Acha95b] has investigated a number of aspects of data broadcast using periodic push including scheduling and client caching [Acha95a], prefetching [Acha96] and integrating push and pull over a broadcast channel [Acha97]. The issue of combining broadcast push and unicast pull is addressed in [Stath97]. The mobility group at Rutgers [Imie94] has done significant work on data broadcasting in mobile environments. A main focus has been on indexing in order to reduce power consumption at the mobile clients. Viswanathan [Vis94] has studied integrating push and pull for a mobile broadcast environment.

VI. Conclusions

In this paper we focused on the challenges of large-scale on-demand data broadcast introduced by high bandwidth broadcasting media such as satellite or cable networks. Unlike previous work, we began by proposing a comprehensive set of performance criteria for scheduling algorithms. These criteria include worst case as well as average response time, three types of scalability, and robustness to changes in the nature and or intensity of the workload. We then described how previous algorithms fail in one or more of these criteria.

We proposed a scheduling algorithm called RxW, that provides a balanced treatment of hot and cold pages in order to achieve a good overall performance. The algorithm uses a novel pruning technique to reduce the search space for making broadcast decisions. While the pruning was shown to be effective, it was observed that such an algorithm could still eventually become a bottleneck for very large applications.

We developed an *approximate*, parameterized variant of RxW that allows the search space to be reduced further, at the possible expense of making less efficient use of the broadcast bandwidth. By varying a single parameter, the algorithm can be tuned from the regular RxW algorithm, to a *constant time* approach that provides maximal scalability. We demonstrated the performance, scalability, and robustness of the different RxW variants through an extensive set of performance experiments.

In terms of future work, we plan to integrate the on-demand scheduling described here, with push-based and other forms of data delivery as part of a larger Dissemination-Based Information Systems (DBIS) framework as described in [Fran97]. We also plan to investigate the scheduling of broadcast for hierarchical broadcast environments.

Acknowledgements

The authors would like to thank to Ugur Cetintemel, Bjorn Thor Jonsson and Mustafa Uysal for their helpful comments on an earlier draft of this paper.

References

[Acha95a] S. Acharya, R.Alonso, M. Franklin, S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments", Proc. ACM SIGMOD Conf., San Jose, CA, 1995.

[Acha95b] S. Acharya, M. Franklin, S. Zdonik, "Dissemination-based Data Delivery Using Broadcast Disks", IEEE Personal Communications, 2(6), 1995.

[Acha96] S. Acharya, M. Franklin, S. Zdonik, "Prefetching from a Broadcast Disk". Proceedings of the International Conference on Data Engineering, New Orleans, LA, Feb 1996.

[Acha97] S. Acharya, M. Franklin, S. Zdonik, "Balancing Push and Pull for Data Broadcast", Proc. ACM SIGMOD, May 1997.

[Aksoy97] D.Aksoy, M.Franklin. "On-Demand Broadcast Scheduling", Technical Report, CS-TR-3854, University of Maryland, 1997.

[Baru96] S.Baruah, A.Bestavros, "Pinwheel Scheduling for Fault-tolerant Broadcast Disks in Real-time Database Systems", Technical Report TR-96-023, Boston University, August 1996.

[Best96] A.Bestavros,,C.Cunha, "Server-initiated Document Dissemination for the WWW", IEEE Data Engineering Bulletin, 1996

[Bowen92] T. Bowen, G. Gopal, K. Lee, A. Weinrib, "The Datacycle Architecture", CACM Vol 32, No 12, December 1992.

[Dao96] S. Dao and B. Perry, "Information Dissemination in Hybrid Satellite/Terrestrial Networks", Data Engineering, 19(13), 1996.

[DirecPC] Hughes Network Systems, *http://www.direcpc.com*, July 97.

[Dyke86] H.D. Dykeman, M. Ammar, J.W. Wong, "Scheduling Algorithms for Videotex Systems Under Broadcast Delivery", IEEE International Conference on Communications, Toronto, Canada, 1986.

[Fran97] M. Franklin, S. Zdonik, "A Framework for Scalable Dissemination-Based Systems", Proc. ACM OOPSLA Conf., 1997.

[Giff90] D. Gifford, "Polychannel Systems for Mass Digital Communication", CACM, 37(10), 1994.

[Herm87] G.Herman, G.Gopal K.Lee, A.Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems", Proc. ACM SIGMOD Conf., San Francisco, CA, May 1987.

[Hybrid] High-Speed Data Access for Teleworkers, *http://www.hybrid.com*.

[Imie94] T. Imeilenski, B.R. Badrinath, "Energy Efficient Indexing On Air", Proc. ACM SIGMOD Conf., Minneapolis, MN, May 1994.

[Intel94] Intel Corporation. Introduction to Intercast Technology, *http://www.intercast.com*, 1994.

[Knuth81] D. Knuth, "The art of Computer Programming - Volume II", Addison-Wesley, 1981.

[Schw86] H.D. Schewtman, "CSIM: A C-based Process Oriented Simulation Language", Proc. of the Winter Simulation Conf., 1986.

[Shekhar96] S. Shekhar, A. Fetterer and D.R. Liu, "Genesis: An Approach to Data Dissemination in Advanced Traveler Information Sytems", Data Engineering, 19(3), 1996.

[Stath97] K. Stathatos, N. Roussopoulos, and J.S. Baras, Adaptive Data Broadcast in Hybrid Networks, in Proc. VLDB, 1997.

[Su97] C.J. Su, L.Tassiulas. "Broadcast Scheduling for Information Distribution", Proc. IEEE INFOCOM, 1997.

[Teledesic] "Application of Teledesic Corporation for a Low Earth Satellite System in the Domestic and International Fixed Satellite Services," filed by Teledesic Corporation with the Federal Communications Commission, March 21, 1994.

[Vaidya96] N.H. Vaidya and S. Hameed. "Data Broadcast in Asymmetric Wireless Environments". Proc. of Workshop on Satellite-based Information Services (WOSBIS), New York, November, 1996.

[Vis94] S.R.Viswanathan, "Publishing in Wireless and Wireline Environments", PhD Thesis, Rutgers University, 1994.

[Wong88] J.W. Wong, "Broadcast Delivery", in Proceedings of IEEE,pp. 1566-1577, Dec. 1988.

[Yan96] T. Yan and H. Garcia-Molina, "Efficient Dissemination of Information on the Internet", Data Engineering, 19(13), 1996.