Evaluation of a Broadcast Scheduling Algorithm

Murat Karakaya and Özgür Ulusoy

Department of Computer Engineering
Bilkent University Ankara, Turkey
{muratk,oulusoy}@cs.bilkent.edu.tr

Abstract. One of the two main approaches of data broadcasting is *pull-based* data delivery. In this paper, we focus on the problem of scheduling data items to broadcast in such a pull-based environment. Previous work has shown that the *Longest Wait First* heuristic has the best performance results compared to all other broadcast scheduling algorithms, however the decision overhead avoids its practical implementation. Observing this fact, we propose an efficient broadcast scheduling algorithm which is based on an approximate version of the Longest Wait First heuristic. We also compare the performance of the proposed algorithm against well-known broadcast scheduling algorithms.

1 Introduction

There exist two main approaches for data dissemination in broadcast systems: *push* and *pull* [1, 2, 12, 19]. In push-based data delivery, the information server tries to predict data needs using the knowledge provided by user profiles or subscriptions. The server constructs a broadcast schedule in which initiation of data transmission does not require an explicit request from mobile users. The server repetitively transmits the content of broadcast schedule to user population. Mobile users monitor the broadcast channel and retrieve the items they require as they arrive. On the other hand, in a pull-based environment, clients explicitly request data items by sending message to the server. The requests are compiled in a service queue, and a scheduling algorithm decides which data item should be broadcast.

In this paper, we focus on the problem of data broadcast scheduling in a pull-based data delivery environment. The main contributions of our work can be described as follows. First, previous work has shown that the *Longest Wait First* (LWF) heuristic has the best performance results compared to all other broadcast scheduling algorithms, however the decision overhead avoids its practical implementation [6, 7, 19, 5]. We propose to use an approximate version of the LWF heuristic which can considerably remove the decision overhead of LWF. Second, the implementation of the approximate heuristic is carefully designed and also parameterized to increase the performance with respect to different criteria. Third, some heuristics proposed to be used in push-based broadcast environments are modified and evaluated in the pull-based broadcast environment we simulate. And, finally detailed simulation tests are conducted and reported.

The remainder of the paper is organized as follows. In Section 2, we introduce a mobile computing environment that we assume in our work and summarize related work. In Section 3, we describe our approximate heuristic and its implementation, *Bucketing* scheduling algorithm. Performance evaluation results of the proposed algorithm are provided and compared with the results of some well-known broadcast scheduling algorithms in Section 4. Finally, in Section 5 concluding remarks are provided.

2 Background

2.1 Mobile Computing Environment

In a common architectural model used for a mobile computing environment [8, 9, 12], geographical area is divided into regions, called *cells*, each of which is covered and serviced by a stationary controller. There exist two types of computers; *mobile units* (computers) (MUs) and *stationary computers* (SCs). SCs are connected together via a fixed network. Some of SCs are equipped with wireless interfaces to communicate with MUs and called *mobile support stations* (MSSs). MSSs behave as entry points from MUs to the fixed network. MUs can consume and also produce information by querying and updating the online database stored on SCs. MSSs can be proxy servers on behalf of the other SCs or they can themselves be information servers.

It is assumed in this mobile environment that there is a single broadcast channel dedicated to data broadcast. Users monitor this channel continuously to get the data items they require. There is a *backchannel* which enables MUs to send data requests to MSSs.

2.2 Related Work

The first work related with broadcasting in a pull-based environment is by Ammar and Wong in the context of *teletext* and *videotext* systems [6, 7, 19]. In [19], Wong proposes three alternative architectures for broadcast information delivery systems: *one-way broadcast (push)*, *two-way interaction (pull)*, and *one-way broadcast/two-way interaction (hybrid)*. The heuristics used in two-way interaction are as follows [19]:

- The well-known FCFS algorithm has been modified in such a way that if a page has been requested and placed in the service queue, a new request for that page is ignored. In this way, redundant broadcasts of the same page are avoided [5].
- Another heuristic proposed to be used in broadcast scheduling is *Most Requested First* (MRF). As the name of the heuristic implies, the page with the largest number of pending requests is selected to broadcast.
- The MRF heuristic is configured to break ties in favor of the page with the lowest request probability if the request probabilities of the pages are available to the scheduling algorithm. This version of the heuristic is termed *Most Requested First Lowest* (MRFL).

- The heuristic which selects the page with the largest total waiting time of all pending requests is *Longest Wait First* (LWF).

These heuristics are evaluated in [19] and it is concluded that when the system load is light, the mean response time is not sensitive to the heuristic used. This is due to the fact that in light loads, few scheduling decisions need to be made. On the other hand, when the system load is high and the page request probabilities follow *Zipf's Law* [20], LWF has the best performance, whereas FCFS has the worst.

Vaidya et al. have worked on data broadcast scheduling algorithms for push-based environments extensively and proposed several scheduling algorithms [10, 11, 13, 14, 17, 18]. In [13], Jiang and Vaidya also investigate how the variance of response time can be minimized. The authors claim that their work and algorithms can be applied to pull-based environments as well. Therefore, we take their algorithms into consideration while devising our heuristic.

The work which is most related to our work is the one performed by Aksoy and Franklin [4, 5]. The authors have focused on pull-based systems and proposed a scheduling algorithm which improves and unifies FCFS and MRF heuristics. They conclude that the LWF heuristic has the best performance results according to overall mean waiting time. However, the authors also point out that the straightforward implementation of LWF is not practical. Aksoy and Franklin suggest to integrate FCFS and MRF in a practical way to combine their advantages and eliminate the disadvantages. As a result, the authors propose the RxW heuristic which balances the selection criterion between the number of pending requests and the first request arrival time of a data item. RxW computes the product of the total number of pending requests (R) and waiting time of the first request (W) of that data item, and selects the data item with the maximum RxW value.

3 Bucketing Algorithm

We have aimed to develop a scheduling algorithm that can minimize both the mean waiting time and its variance, as well as is robust to changes in mobile environment and has lower overhead. We describe a new heuristic that we name *Approximate Total Waiting Time* (ATWT). The proposed ATWT heuristic is implemented using a bucketing scheme and the resulting algorithm is termed *Bucketing Algorithm.*

3.1 Approximate Total Waiting Time

In order to decrease the amount of computation, we first assume that all requests for a page come at the same time as the first one. Thus, we only keep the arrival time of the first request for each page. When we need to compute total waiting time of a page, we can simply multiply the number of pending requests with the elapsed time since the first request arrived. This approximation gives us the upper bound of total waiting time of a page. Provided that requests arrival is governed by the Poisson process, if a page is broadcast τ time units after the arrival of the first request to it, mean waiting time for pending requests for this page is $\frac{\tau}{2}$ [6, 17]. This fact gives an approximation to compute total waiting time for a page as follows:

$$W_p(t) = \frac{t - A_p}{2} * R_p(t) \tag{1}$$

where t is the current time, A_p is the first request arrival time, and $R_p(t)$ is the total number of pending requests for page p at time t. $W_p(t)$ is the approximate total waiting time for page p. The LWF heuristic needs to compute the total waiting time for every page to select the one with the largest value. We can drop the division by 2 in (1) to simplify the calculation since $W_p(t)$ of each page will be compared. This finalizes the basic formulation, that we call *Approximate Total Waiting Time* (ATWT). ATWT enables us to record less information and do less computation [1].

Finding the Maximal ATWT. The direct implementation of the heuristic we propose above has a time complexity of $O(N)$, where N is the total number of requested pages. In order to avoid the calculation of each requested page's ATWT, we use a method which selects a few pages and calculates only their ATWTs to select the page with *maximal ATWT* value. Our implementation is based on a bucketing technique. We classify the pages according to the number of pending requests associated with them. All the pages that lie in bucket i will have pending request numbers ranging between 2^{i-1} and 2^i-1. The number of buckets is limited by the number of pending requests for distinct pages. There will be $\lceil \log(R+1) \rceil$ buckets of pages, where R is the number of pending requests of the most requested page in the system. In each bucket, the pages are ordered according to their first request arrival time. The first page of each bucket is the first requested page within that bucket.

Whenever we need to find the page with the maximal ATWT, we compare only the ATWT values of the first pages of each bucket. Since the number of buckets is logarithmic with respect to the most requested page's request number, we would examine very few candidates. The page with the largest ATWT value is selected among the first entries of all buckets. It can be shown that the bucketing scheme results in selecting a page with an ATWT value which is at least half of the maximum ATWT value.

[1] A formula similar to the approximation we provide for total waiting time has also been suggested by Aksoy and Franklin [5] but through completely different reasoning and observations from ATWT.

3.2 Implementation of Bucketing Algorithm

The data structure used for each requested page in Bucketing algorithm is illustrated in Figure 1. Each bucket is a linked list of requested pages [2]. Pages are ordered in the linked lists according to the first request arrival time. Fields *Prev* and *Next* are pointers to the previous and next pages, respectively in the linked list.

Fig. 1. Page data structure

Prev	A First Request Arrival Time	Next
Previous page according to A value	R Total number of pending requests	Next page according to A value

Entries for pages are placed in buckets by mapping total number of requests to bucket number. A page with a total request number i is placed to bucket $\lfloor \log(i) \rfloor + 1$.

Bucketing algorithm works as follows: when a request arrives to the server, if it is the first request for the page, its arrival time is recorded to field A of the page data structure and the number of pending requests (R) is set to one. The page is placed at the end of the linked list in the first bucket since its R value is 1.

Otherwise, if the page was requested and not yet broadcast, R is incremented by one. Then, if the page does not belong to the existing bucket anymore, it is moved to the appropriate bucket according to its R value. The page is then inserted in the linked list of this bucket with respect to its A value.

In the selection of the page to broadcast, only the first page of each bucket is examined. The page with the largest ATWT is broadcast and removed from the bucket. The bucketing scheme reduces the decision overhead considerably without deteriorating the quality of the produced broadcast.

We have also implemented a variant of Bucketing algorithm, called *k-depth* Bucketing algorithm, in which we examine the first k entries of each bucket. By comparing more entries in a bucket, it is expected to have more accurate ATWT.

Minimizing the Variance of Waiting Time. We have investigated the variance of waiting time produced by ATWT and several other heuristics. In [13], Jiang and Vaidya reformulate the algorithm presented in [17] considering variance metric and propose a new algorithm called α-*algorithm*. The authors also

[2] For performance concerns, instead of using a linked list data structure, a heap data structure can also be implemented to store the items in each bucket. However, for the sake of simplicity we prefer to implement a linked list data structure in the simulation.

Table 1. Simulation parameters

Symbol	Description	Default	Range	Unit
λ	Mean Req.Arrival Rate	10	[10-100]	req./tick
Θ	Request Pattern Skewness	1.0	[0.1-1.0]	-
dbSize	Database Size	1,000	[1,000-10,000]	pages
pSize	Page Size	1	-	tick

claim that the algorithm can be adapted to pull-based systems as well. Therefore, we modify the computation of ATWT in our heuristic in a way similar to that suggested in [13] as follows:

$$(t - A_p)^{\alpha} * R_p(t) \tag{2}$$

where α can be assigned different values in order to tune variance of waiting time and mean waiting time.

4 Simulation Results

We have simulated the mobile environment introduced in Section 2.1 and conducted extensive experiments to evaluate the performance of Bucketing algorithm relative to other scheduling algorithms. The simulation program was written in CSIM [16]. Due to lack of space we can not provide all the experiments and their results. For more details please refer to [15].

4.1 Simulation Model

Our simulation model consists of three main components: a *mobile support station* (MSS), a population of *mobile units* (MUs) and *communication channels*. Published requests are kept in *service queue*. *Online database* stores the shared data items. The decision process is performed by a *scheduling algorithm*. Client population represents MUs within the cell. Communication channel is a two-way medium. In *broadcast channel*, selected data items are delivered to MUs, whereas *backchannel* is used to send data requests of MUs to MSS.

Simulation parameters and their values are summarized in Table 1. We assume that MSS is a information server and it serves for the demands of MUs from database located on itself. *dbSize* is the total number of available data items at an *MSS*. Data items are numbered from 1 to *dbSize*, where a data item is, for example, a web page or a file. We use the terms *data item* and *page* interchangeably since the information server can be a database or web server.

Requests of MUs are represented by a single request stream. Request arrivals are assumed to be *Poisson* with a mean value of λ. By increasing λ, we can simulate a higher system load. MUs may exhibit data locality, querying a particular subset of the database repeatedly [9,12]. This subset is a *hot spot* for an MU.

In general, a user may request multiple items simultaneously and would expect to receive mutually consistent versions of the requested items. In this paper, similar to many of the past work, we consider the case where a user demands only one item per request, and unless the user gets the item, a new request is not initiated. In our work, the effect of transmission errors is not considered. We assume that when a data item is broadcast, all the users requesting that item receive it completely.

It is assumed that access probabilities follow the *Zipf* [20] distribution over the database items as in many other related work (e.g., [3, 5, 19]). Data items are supposed to be ordered in the database according to their access probabilities in decreasing order, i.e., the most favorite data item is in the first place in the database. Zipf's law states that the relative probability of a request for the i'th most popular data item is proportional to $\frac{1}{i}$, where i is between 1 and dbSize. The Zipf distribution can be formulated to show the demand probability of each data item as below:

$$p_i = \frac{(1/i)^{\Theta}}{\Sigma_{i=1}^{dbSize}(1/i)^{\Theta}} \tag{3}$$

where Θ is a parameter termed *access skew coefficient* [18]. By changing the value of Θ, different Zipf distributions can be obtained. For instance, if Θ is set to zero, Zipf distribution turns out to be a uniform distribution with p_i=1/dbSize.

The time to broadcast a data item is calculated by a specific time unit called *tick*. We assume that page sizes (*pSize*) are equal and each page can be transmitted in one tick. The use of tick as a time unit enables us to compare easily the results of systems with different properties such as bandwidth and data item size. Mean waiting time is measured by the number of ticks.

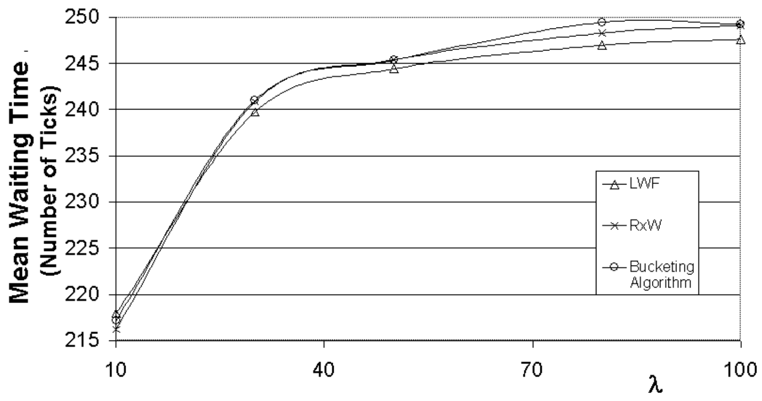
For evaluating a broadcast scheduling algorithm for a particular set of parameters, the broadcast schedule is produced for at least 30,000 cycles. Furthermore, we run each configuration ten times and use the averages as final estimates.

4.2 Performance Criteria

We have used the following performance criteria in our evaluations:

- *Waiting time of a request* is defined as the duration from when the request is made until the desired data item begins to be transmitted on the channel.
- *Variance of waiting time* is taken into consideration to evaluate the Quality of Service experienced by any user [13], where the overall mean waiting time is an indication of the idle time for the whole user population.
- *Worst waiting time* is defined as the maximum amount of time that any user request waits before being satisfied. The reason to use this criterion is to check if the algorithm causes starvation of some requests, which is an important property for interactive applications [5].
- *Decision overhead* is the time taken by the computation which should be done for selecting a data item to broadcast next. The decision overhead of a good scheduling algorithm should not be high.

Fig. 2. Mean waiting times of the LWF, RxW, and Bucketing algorithms



4.3 Mean Waiting Time

In this experiment, mean request arrival rate (λ) is varied from 10 requests to 100 requests per tick. As depicted in Figure 2, mean waiting times for all algorithms are increasing while the request arrival rate is getting higher. However, after a certain rate, it levels off. In the figure, the results of MRF and FCFS are not presented. Because they have much larger mean waiting time than the other algorithms. LWF, RxW, and Bucketing algorithms are characterized by almost the same mean waiting time. The largest difference between any two of these three algorithms is not more than 0.8%.

Although LWF is a good algorithm in terms of mean waiting time, straightforward implementation of it is not practical for large databases and high-speed broadcast channels. RxW and Bucketing algorithm are the only algorithms which are practical to implement and satisfactory in terms of mean waiting time results.

We have conducted several simulation tests to record the impact of the different database sizes and access skewness values. It is observed that the resulting waiting times of algorithms are relatively almost the same. As the database size increases, the mean waiting time increases as well. There is not much difference between the scalability of the algorithms with respect to database size. Therefore, concerning the time required to perform the simulation experiments, we preferred to use a default database size of 1,000 pages for all other experiments without losing generality.

As the skewness of the Zipf distribution is increased, mean waiting time values of the algorithms, except FCFS, are getting considerably smaller. This result is due to the fact that the highly skewed request distribution (i.e., $\theta \geq 0.7$) leads to the existence of many pending requests to a few data items, and that broadcasting one of the most requested data items satisfies many pending requests. RxW, LWF and Bucketing algorithms take the number of pending requests into account and this property causes more efficient use of the broadcast channel. However, when θ is 0.1, the distribution reduces to an almost uniform distribution, and each data item has almost the same pending requests. In that case, all the scheduling algorithms lead to almost the same mean waiting time. FCFS does not consider the pending request number in broadcast scheduling. Hence, the mean waiting time obtained with this algorithm does not improve much when the access skewness increases.

4.4 Variance of Waiting Time

In Section 3.2, we have modified our algorithm to handle the trade-off between mean waiting time and variance of waiting time. The ATWT value used as a selection criterion in Bucketing algorithm has been modified as in (2).

To observe the effect of different α values both on variance of waiting times and mean waiting time of Bucketing algorithm we conduct several experiments. In these experiments, α parameter of (2) is varied from 0.5 to 3.0, while using the other default parameter values given in Table 1. We observe that the trade-off between mean waiting time and variance of waiting time is evident. For higher values of α, the variance is improving, on the other hand, the mean waiting time of the algorithm is getting worse. The mean waiting time of the algorithm improves while α is increased from 0.5 to a certain value, which is 0.9 in our experiment. Then, for the values larger than this threshold, the mean waiting time begins to worsen again. This result is due to the fact that for the α values higher than 1, the modified ATWT heuristic in (2) attaches more importance to the waiting time of the first request than the total number of pending requests of a page. Therefore, the heuristic selects the pages similar to those selected with FCFS. On the other hand, when α is set to values lower than 1, the heuristic behaves in favor of the most requested pages like MRF. As a result, as the α value gets smaller or larger than 1, the experienced mean waiting time becomes more similar to that of one of the two algorithms.

We also conducted an experiment to compare the variance obtained with all scheduling algorithms. The results depicted in Figure 3 show that FCFS has the lowest degree of variance due to the fact that in the worst case, the algorithm broadcasts any requested data item after broadcasting the whole set of data items in the database. That is, for the waiting time of a request, there is an upper bound which is determined by the database and page sizes. This upper bound also limits the variance of the waiting time. However, we can not claim this argument for the other algorithms.

The performance results obtained after modifying the computation of ATWT in our algorithm as in (2) are presented in Figure 3. In this experiment, we set the α value to 2. For the high workloads, the modified ATWT has even better variance of waiting time than that of FCFS. However, as discussed above, the mean waiting time of the modified ATWT has become slightly worse (see Figure 4). With a greater value for α (e.g., 3), the variance of waiting time can be further decreased; nonetheless, the the mean waiting time would become worse.

Fig. 3. Comparing Bucketing algorithm when α=2

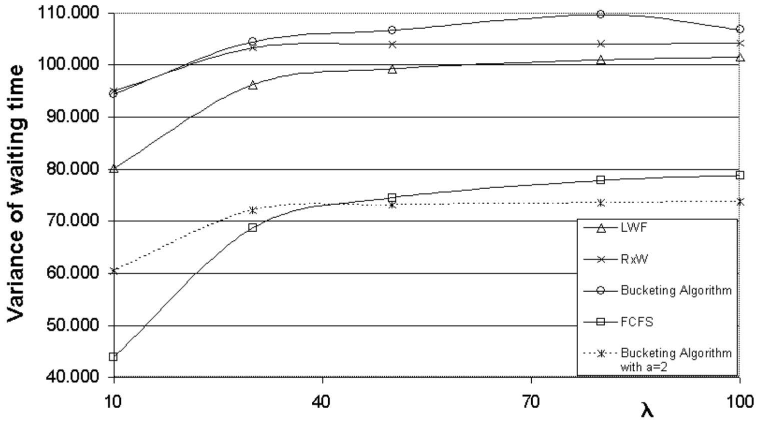


Fig. 4. Mean waiting time of Bucketing algorithm with α=2

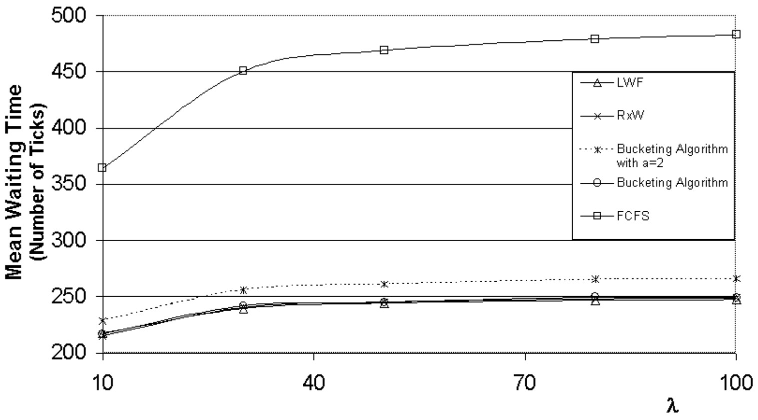
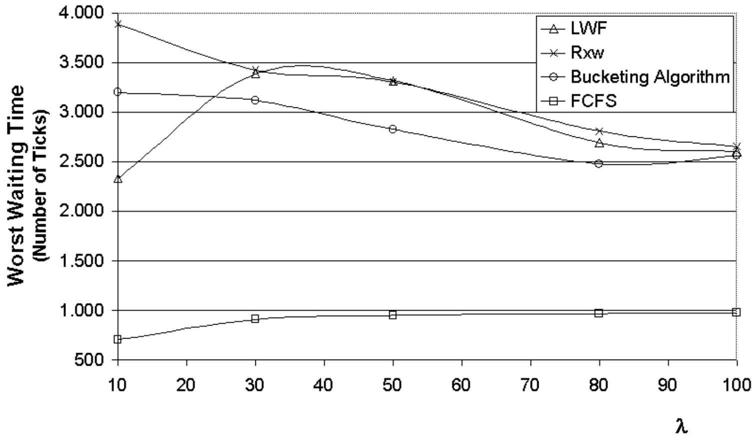


Fig. 5. Worst waiting time



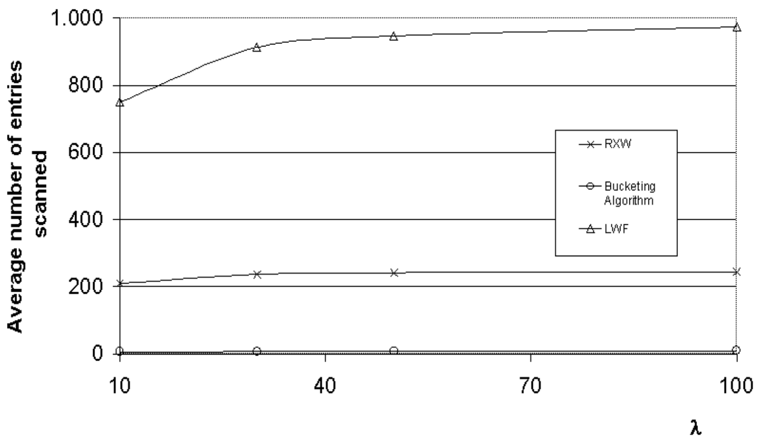
4.5 Worst Waiting Time

The reason to investigate the worst waiting time is to check if a scheduling algorithm causes starvation of any request, which is an important property that should be avoided in interactive applications. Figure 5 displays the results for the longest waiting time experienced by any MU during the whole simulation time. For the default values of the simulation parameters presented in Table 1, FCFS has the lowest worst waiting time among all the algorithms. As discussed above, when FCFS is employed as the scheduling algorithm, any requested data item will be broadcast after the data items previously requested and the number of these data items is limited by the database size. In other words, the largest possible worst waiting time of a request is the time taken by broadcasting all the database items. However, for the other algorithms, it might be possible that a request waits while some of the data items are broadcast multiple times. Bucketing algorithm has considerably lower worst waiting time values compared to RxW. The results obtained with MRF are so much larger than those of the other algorithms that we do not include them in the figure.

4.6 Scheduling Decision Overhead

As discussed previously, a good scheduling algorithm should not have much scheduling decision overhead. In implementing the performance model, the decision overhead associated with each algorithm has not been considered. Since this overhead can not be accurately simulated for different types of algorithms, the time spent during the decision process has been ignored. For a comparative evaluation of decision overhead of the scheduling algorithms, we examined the number of requests scanned for selecting one of them to broadcast. If the number is large, the decision takes much more time and may become a bottleneck.

Fig. 6. Decision overhead



We compared the average number of data items scanned by three algorithms: LWF, RxW and Bucketing. FCFS is not included in this experiment. It just broadcasts the request that has arrived first, and does not need to compare any entry. On the other hand, its overall waiting time is so bad that it is not a competitive algorithm to be used.

As depicted in Figure 6, LWF has the highest decision overhead while Bucketing algorithm has the lowest. The overhead of RxW is in between. Compared to other algorithms, Bucketing algorithm examines significantly fewer number of requests at each scheduling decision. For a request rate of 10 requests per tick, LWF compares 130 times more entries and RxW compares 36 times more entries than that of Bucketing algorithm [3].

4.7 Improving the Bucketing Algorithm

We have tried to improve the mean waiting time of our Bucketing algorithm and implemented the *depth* approach as presented in Section 3.2.

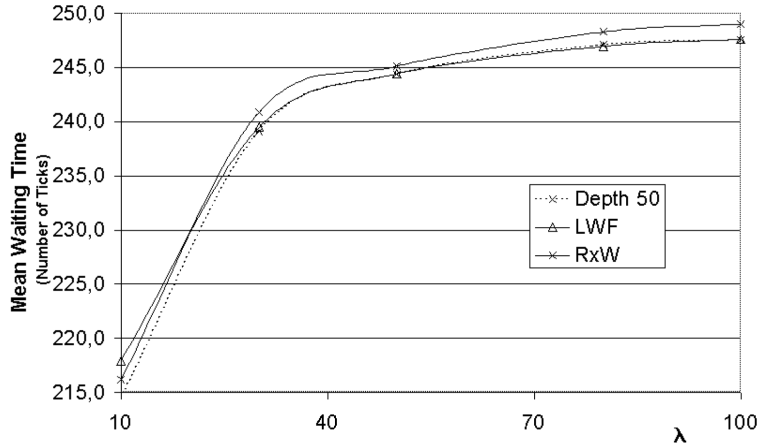
There is a trade-off between the decision overhead and the mean waiting time in this approach. As we increase the search depth, we need to compare more entries of ATWT values, and we obtain lower mean waiting time (see Table 2). When we set the depth parameter to 50, the resulting mean waiting time of Bucketing algorithm is less than that of RxW as can be seen in Figure 7.

[3] In [5], approximate versions of the RxW algorithm are proposed and they are shown to lead to less comparisons in deciding which data item to broadcast. However, these approximate versions have worse mean waiting time compared to RxW. The Bucketing algorithm, as discussed in Section 4.3, produces almost the same mean waiting times as RxW, while leading to less scheduling decision overhead.

Table 2. Effect of using depth parameter on mean waiting time

Depth	Mean Waiting Time
50	214.1
25	215.6
10	216.3
1	217.5

Fig. 7. Bucketing algorithm with depth=50 and the other competitive algorithms



5 Conclusion

In this paper, the problem we attack is the design of a broadcast scheduling algorithm which efficiently meets the demands of a mobile computing environment and mobile users. We have first proposed a new broadcast scheduling heuristic, ATWT, which is an approximate version of LWF heuristic [19]. Then, we have developed an algorithm called Bucketing algorithm to implement ATWT heuristic by using a bucketing scheme. Finally, we have conducted extensive simulation experiments to evaluate the performance of our algorithm, and to compare the performance results against those of previously proposed scheduling algorithms.

Considering the performance results, the first remark to be done is that the most competitive algorithm to our algorithm is RxW [5]. The other algorithms, except LWF, do not produce good results with respect to the main performance criterion, the *overall mean waiting time*. Although, LWF has better performance than all the others, it has serious drawbacks which prevent its practical usage. In terms of the other performance metrics, i.e., *variance of waiting time* and *worst waiting time*, the performance of Bucketing algorithm is better, in general, than that of all other scheduling algorithms.

References

1. Acharya, S., Alonso, R., Franklin, M., Zdonik, S.: Broadcast disks: Data management for asymmetric communication environments. Proceedings of ACM SIGMOD (1995) 199–210
2. Acharya, S., Franklin, M., and Zdonik, S.: Balancing push and pull for data broadcast. Proceedings of ACM SIGMOD Conference. Tuscon, Arizona. (1997)
3. Acharya, S., M., Franklin, J., Zdonik, S.: Dissemination-based data delivery using broadcast disks. IEEE Personal Communications. **2(6)** (December 1995) 50–60
4. Aksoy, D., Franklin, M.: Scheduling for large-scale on-demand data broadcasting. Proceedings of the IEEE INFOCOM Conference. (1998) 651–659
5. Aksoy, D., Franklin, M.: Rxw: A scheduling approach for large-scale on-demand data broadcast. ACM/IEEE Transactions on Networking. **7** (1999) 846–860
6. Ammar, M., H., Wong, J., W.: The design of teletext broadcast cycles. Performance Evaluation. **5** (November 1985) 235–242
7. Ammar, M., H., Wong, J., W.: On the optimality of cyclic transmission in teletext systems. IEEE Transactions on Communications. **35** (January 1987) 68–73
8. Barbara, D.: Mobile computing and database: A survey. IEEE Transactions on Knowledge and Data Engineering. **11** (January-February 1999) 108–117
9. Barbara, D., Imielinski, T.: Sleepers and workaholics: Caching strategies in mobile environment. ACM SIGMOD RECORD. **23** (May 1994)
10. Hameed, S., Vaidya, N.,H.: Log-time algorithms for scheduling single and multiple channel data broadcast. Proceedings of the Third Annual ACM/IEEE International Conference on Mobile Computing and Networking. Budapest, (September 1997) 90–99
11. Hameed, S., Vaidya, N.,H.: Efficient algorithms for scheduling data broadcast. Wireless Networks. **5** (1999) 183–193
12. Imielinski, T., Badrinath, B., R.: Mobile wireless computing: Challenges in data management. Communications of the ACM (October 1994) 19–27
13. Jiang, S., Vaidya, N., H.: Response time in data broadcast systems: Mean, variance and trade-off. Proceedings of International Workshop on Satellite-based Information Services (WOSBIS). (October 1998)
14. Jiang, S., Vaidya, N., H.: Scheduling data broadcasting to "impatient" users. Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access. Seattle, WA USA, (August 1999) 52–59
15. Karakaya, M., Ulusoy, Ö.: An efficient broadcast scheduling algorithm for pull-based mobile environments. Submitted to the IEEE/ACM Transactions on Networking.
16. Schwetman, H.: Csim18 the simulation engine. In J., Charnes, D., Morrice, D., Brunner, J., Swain, editors. Proceedings of the 1996 Winter Simulation Conference. San Diego, CA, (1996) 517–521
17. Vaidya, N., H., Hameed, S.: Scheduling data broadcast in asymmetric communication environments. Wireless Networks. **5** (1999) 171–182
18. Vaidya, N., H., Jiang, S.: Data broadcast in asymmetric wireless environments. Proceedings of First International Workshop on Satellite-based Information Services (WOSBIS). NY, (November 1996)
19. Wong, J.,W.: Broadcast delivery. Proceedings of The IEEE. 76, (December 1988) 1566–1577
20. Zipf, G.,K.: Relative frequency as a determinant of phonetic change, volume,XL. Reprinted from the Harvard Studies in Classical Philiology. (1929)