COMBINATORIAL OPTIMIZATION IN COMMUNICATION NETWORKS
Maggie Cheng, Yingshu Li and Ding-Zhu Du (Eds.) pp. 622 - 637

Stretch-Optimal Scheduling for On-Demand Data Broadcasts

Yiqiong Wu, Jing Zhao, Min Shao, and Guohong Cao
Department of Computer Science & Engineering
The Pennsylvania State University, PA 16802
E-mail: {ywu,jizhao,mshao,gcao}@cse.psu.edu

1 Introduction

For dissemination-based applications, unicast data delivery as used by the current web servers may not be scalable. With unicast, a data item must be transmitted for each client that have made the request, and hence the load on the server and the network increases with the number of clients. Broadcasting techniques have good scalability since a single broadcast response can potentially satisfy many clients. Recent technology advances in satellite networks, cable networks, wireless LANs and cellular networks make it possible to disseminate information through broadcasting.

A key consideration in designing broadcast systems is the algorithm used to schedule the broadcast. Scheduling algorithms have been extensively studied in the context of operating systems [11]. Traditional scheduling algorithms such as first come first served (FCFS), round robin, shortest job first, are used in processor scheduling and disk scheduling. These scheduling algorithms are used to optimize the response time, throughput, or fairness. Since most of these scheduling algorithms are designed for the point-to-point communication environments, they may not be applicable to the broadcasting environments.

A number of researchers addressed issues on broadcast schedule. For example, Wong [14] studied several scheduling algorithms such as first-come-first served (FCFS), longest wait time (LWT), most requests first (MRF) in broadcasting environments. Researches in [5, 8] investigated techniques to index the

broadcast data to save power in mobile environments. Su and Tassiulas [12] formulated the broadcast scheduling as a dynamic optimization problem and proposed efficient suboptimal solutions which can achieve a mean access latency close to the lower bound. Hameed and Vaidya [7] applied existing fair queueing algorithms to broadcast scheduling. Most of the research [2, 6, 12] on broadcast scheduling focus on *push-based* broadcasting; that is, the server delivers data using a periodic broadcast program based on precompiled access profiles and typically, without any active user intervention. The application of such schemes are limited since they are not capable of adapting to the dynamic user requirements.

There is another kind of broadcasting: *on-demand (pull-based)* broadcasting, where a large and dynamic client population request data from an information server which broadcasts data to the clients based on these requests. Aksoy and Franklin [3] initiated the study of on-demand broadcast scheduling. They proposed a $R \times W$ scheduling algorithm which provides balanced treatment of both hot and cold data resulting in a good overall performance. The algorithm combines MRF and FCFS, and uses a novel pruning technique to reduce the computation overhead. However, the $R \times W$ algorithm assumes that each data item has the same data size. Hence, it is not suitable for requests with variable data size, because response time alone is not a fair measurement for requests with different data size. Acharya and Muthukrishnan [1] addressed the broadcast scheduling problem in heterogeneous environments, where the data have different sizes. The solution is based on a new metric called $Stretch$, defined as the ratio of the response time of a request to its service time. Based on stretch, they proposed a scheduling algorithm, called longest total stretch first (LTSF) to optimize the stretch and achieve a performance balance between worst cases and average cases. A straightforward implementation of LTSF is not practical for a large system, because the server has to recalculate the total stretch of each data item with pending requests at each broadcast interval to decide which data to broadcast next.

In this chapter, we propose a stretch optimal scheduling algorithm for on-demand broadcast systems based on the observation that LTSF is not optimal in terms of the overall stretch. One nice property of the proposed algorithm is that it is extremely simple and the computation overhead is significantly lower compared to LTSF. Analytical studies as well as detailed simulation experiments are used to verify these claims. Simulation results show that our algorithm can significantly reduce the overall stretch and it still has a comparative worst-case system response time compared to existing scheduling algorithms.

The rest of the chapter is organized as follows. Section 2 develops necessary background. In section 3, we present the stretch optimal scheduling

algorithm, develop analytical models to demonstrate its optimality in terms of stretch, and address some implementation issues. Section 4 evaluates the performance of the proposed algorithm. Section 5 concludes the chapter.

2 Preliminaries

In this section, we describe our system model, the performance metrics and the related scheduling algorithms.

2.1 The System Model

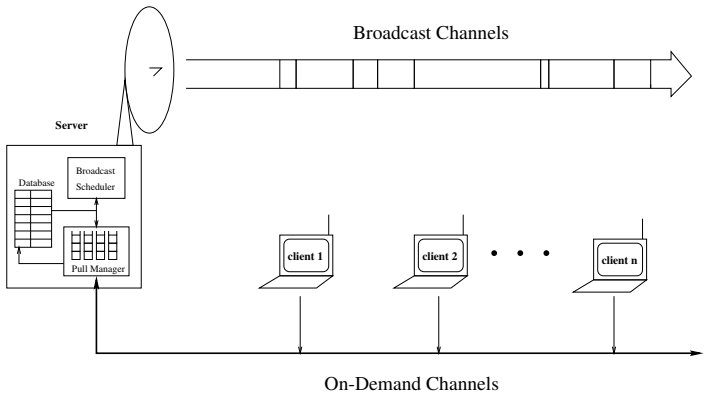

Figure 1: An on-demand broadcast system

Figure 1 illustrates a broadcast environment. which consists of a number of clients. These clients use uplink channels to send requests to the server, which uses a downlink (broadcast) channel to broadcast the responses. In the context of Hughes' DirecPC architecture [13], the uplink channel would be phone lines from the clients to the DirecPC office and the downlink would be the satellite channel. In cellular networks that provide general packet radio service (GPRS) [9] service, the uplink would be the packet random access channel, and the downlink would be the packet broadcast channel.

Similar to previous work on broadcast scheduling [3, 12, 14], we make several assumptions. There is a single broadcast channel monitored by all clients and the channel is fully dedicated to data broadcast. When a client needs a data item that it cannot find locally, it sends a request for the data to the server. These requests are queued up at the server upon arrival. Based on the scheduling algorithm, the server repeatedly chooses an item based on these requests,

broadcasts it over the satellite link, and removes the associated requests form the queue. Clients continuously monitor the broadcast channel after having made the requests. We do not consider the effects of transmission errors. After an item is broadcast, it is received by all clients that are waiting for it. Unlike some previous work [3, 12, 14], we do not assume that the data items have equal sizes.

2.2 Performance Metrics

One popular performance metric is the response time of a request, i.e., the time between sending a request and receiving the reply by a client. In heterogeneous settings, the response time is not a fair measure given that individual requests significantly differ from one another in their service time. In this chapter, we adopt an alternate performance measure, namely the stretch [1, 16, 15] of a request, defined to be *the ratio of the response time of a request to its service time*. Note that the service time is the time to complete the request if it were the only job in the system. The rationale for this choice is based on our intuition; i.e., clients with larger jobs should be expected to be in the system longer than those with smaller requests. The drawback of minimizing response time for heterogeneous workloads is that it tends to improve the system performance of large jobs (since they contribute the most to the response time). Minimizing stretch, on the other hand, is more fair for all job sizes.

We use the average stretch to measure the overall system performance. Since it is important to ensure that the scheduling algorithm does not starve the requests of unpopular data, we also measure the worst-case wait time, which is the maximum amount of time that a client request waits in the service queue before being served. To make a scheduling decision, there is a *decision overhead*. When the database size is very large, the decision overhead may be significant. As a result, some broadcast bandwidth may be wasted since the server spends a large amount of time on making scheduling decisions. Thus, we also measure the decision overhead.

2.3 Related Scheduling Algorithms

Some of the scheduling algorithms related to our work are listed as follows and are used for comparison.

- First Come First Served (FCFS): Broadcasts the data items in the order they are requested. To avoid redundant broadcasts, requests for data items that already have entries in the queue are ignored.

- Longest Waiting Time (LWT): Selects the data for which the total waiting time of pending requests is the largest, i.e., sum of all request's waiting time.

- Most Request First (MRF): The data items are broadcast in the order of the number of pending requests. The item with the most number of pending requests is chosen for broadcast.

- Shortest Service Time First (SSTF): The data items are broadcast in the order of the data service time. The data service time is relative to the data size. The data item with the smallest size is chosen for broadcast.

- Requests times Wait ($R \times W$): The data items are broadcast in the order of the value of function $R \times W$, where R represents the number of pending requests and W represents the first unserved request waiting time. The item with the largest $R \times W$ is chosen for broadcast. The $R \times W$ algorithm is a combination of MRF and FCFS in a way that ensures good scalability and low decision overhead.

- Largest Total Stretch First (LTSF): The data items are broadcast in the order of the total current stretch, i.e., the sum of the current stretches of all pending requests for the data. The data item with the longest total current stretch is chosen for broadcast.

3 The Stretch Optimal Scheduling Algorithm

In this section, we first use an example to show that the LTSF algorithm is not optimal in terms of the overall stretch. Then, we propose a stretch optimal scheduling algorithm and discuss some implementation issues.

3.1 An Example

In LTSF, the server maintains a queue Q_i for each data item i. Since there may be multiple requests for the same item i, let $t_{k,i}$ denote the arrival time of the k^{th} request of item i. s_i denote the data size of item i, and B denote the broadcast bandwidth. Suppose the current time is t. Broadcasting item i at time t has the following stretch.

$$Stretch(i) = \sum_{k \in Q_i} \frac{t + \frac{s_i}{B} - t_{k,i}}{\frac{s_i}{B}} \tag{1}$$

The LTSF algorithm selects the data item with the maximum stretch (based on Equation 1) to broadcast. Although it is intuitively correct in a point-point communication environment, it may not be optimal in an on-demand broadcasting environment. For example, suppose a database has two data items such that $s_1 = 1$, $s_2 = 2$. Assume that the request queues of these items are: $Q_1 = \{t_{1,1} = 8\}$, $Q_2 = \{t_{1,2} = 5, t_{2,2} = 9\}$. For simplicity, let $B = 1$. At time $t = 10$, the server needs to decide which item to broadcast first. If the LTSF algorithm is used:

$$Stretch(1) = \frac{10 + 1 - 8}{1} = 3$$
$$Stretch(2) = \frac{10 + 2 - 5}{2} + \frac{10 + 2 - 9}{2} = 5$$

As a result, the second item is broadcast first, and then the first item is broadcast at $t = 12$ (broadcasting the second item needs 2 time units). Then, the overall stretch is:

$$5 + \frac{12 + 1 - 8}{1} = 10$$

However, if the first item is broadcast first, and the second item is broadcast at $t = 11$. Then, the overall stretch is:

$$3 + \frac{11 + 2 - 5}{2} + \frac{11 + 2 - 9}{2} = 9$$

Thus, the LTSF algorithm cannot minimize the overall stretch.

3.2 The Stretch Optimal Scheduling Algorithm

Before presenting our stretch optimal scheduling algorithm, we first introduce a broadcast scheduling function for item i:

$$S(i) = \sum_{k \in Q_i} \frac{1}{s_i^2} \tag{2}$$

The scheduling algorithm, which selects the data item with maximum $S(i)$ (Equation 2) to broadcast, can minimize the overall stretch.

Proof. We give the sketch of the proof. Based on Equation 1, for any two data items i and j, the scheduling algorithm should determine to broadcast which one first. If the server broadcasts item i first, the overall Stretch is:

$$Stretch_{i \to j} = \sum_{k \in Q_i} \frac{t + \frac{s_i}{B} - t_{k,i}}{\frac{s_i}{B}} + \sum_{k \in Q_j} \frac{t + \frac{s_i}{B} + \frac{s_j}{B} - t_{k,j}}{\frac{s_j}{B}}$$

Similarly, if item j is broadcast first, the overall Stretch is:

$$Stretch_{j\to i} = \sum_{k\in Q_j} \frac{t+\frac{s_j}{B}-t_{k,j}}{\frac{s_j}{B}} + \sum_{k\in Q_i} \frac{t+\frac{s_j}{B}+\frac{s_i}{B}-t_{k,i}}{\frac{s_i}{B}}$$

We should minimize the overall stretch. Let

$$Stretch_{i\to j} - Stretch_{j\to i} = \sum_{k\in Q_j} \frac{s_i}{s_j} - \sum_{k\in Q_i} \frac{s_j}{s_i}$$

To minimize the overall stretch, if $Stretch_{i\to j} - Stretch_{j\to i} < 0$, item i should be broadcast first; otherwise, item j should be broadcast first. Thus, in order to broadcast item i first,

$$Stretch_{i\to j} - Stretch_{j\to i} < 0 \Longrightarrow \sum_{k\in Q_j} \frac{s_i}{s_j} < \sum_{k\in Q_i} \frac{s_j}{s_i} \Longrightarrow \sum_{k\in Q_j} \frac{1}{s_j^2} < \sum_{k\in Q_i} \frac{1}{s_i^2} \tag{3}$$

To minimize the overall stretch, item i is broadcast when $\sum_{k\in Q_i} \frac{1}{s_i^2}$ has the maximum value. □**An example:** Suppose a database has three data items such that $s_1 = 1$, $s_2 = 2$, and $s_3 = 3$. Assume that the request queues of these items are: $Q_1 = \{t_{1,1} = 8\}$, $Q_2 = \{t_{1,2} = 9, t_{2,2} = 9, t_{3,2} = 9, t_{4,2} = 9, t_{5,2} = 9\}$, $Q_3 = \{t_{1,3} = 2, t_{2,3} = 9\}$. At time $t = 10$, the server needs to decide which item to broadcast first. If Theorem 3.2 is followed, since $S(1) = 1$, $S(2) = \frac{5}{4}$, and $S(3) = \frac{2}{9}$, the data broadcasting order is 2-1-3, which has an average stretch of 1.81. The scheduling order of other algorithms and their average stretch costs are listed in Table 1.

Table 1: An example

Scheduling algorithm	Broadcasting Order	Average Stretch
SSTF	1-2-3	1.875
	1-3-2	2.62
Our	2-1-3	1.81
MRF	2-3-1	2.1
FCFS	3-1-2	2.94
LWT, RxW, LTSF	3-2-1	2.88

The algorithm: Based on Equation 2, the overall stretch can be optimized. However, due to the removal of the time parameter in Equation 2, some cold data items may never be served, and the algorithm may suffer from the starvation problem. To address this problem, we add a scheduling deadline rule. When the first request for a data item i arrives, a scheduling deadline is assigned to this pending request. When the deadline passes, the data item has the highest priority to be broadcast. The stretch optimal algorithm is as follows.

1. Scan the request queue of each data item to check if any request has reached the scheduling deadline. If the request for item i has reached the scheduling deadline, let $v = i$ and go to Step 3 directly. If multiple entries have passed their scheduling deadline, use their $S(i)$ to break the tie.

2. Scan the request queue of each data item to find the data item with maximum $S(i)$, and set $v = i$. If multiple entries have the same $S(i)$, use their request arrival time to break the tie.

3. Broadcast item v.

Even with the tie breaking rules, it is possible that multiple candidates have passed their scheduling deadline, and they have the same S value. In this case, the server just randomly picks one.

We observe some interesting properties of the stretch optimal scheduling algorithm.

- For two requested data items with the same number of pending requests, the one with smaller data size should be chosen for broadcast. Hence, if there are similar number of pending requests for each data item, the proposed algorithm has similar performance to SSTF.

- When the requested data items have the same size, the one with more pending requests should be chosen for broadcast. Hence, when the data sizes are similar, the proposed algorithm has similar performance to MRF.

3.3 Implementation Issues

To implement the stretch optimal scheduling algorithm, the server maintains two service queue data structures: a S-Heap, and a D-Link, which are shown in Figure 2. The service queue structure contains a single entry for each data item that has outstanding requests. In addition to an ID, each entry contains S, $S(i)$,

and $1stARV$ which is the arrival time of the oldest outstanding request for the data item. When a request arrives at the server, a hash lookup is performed on the ID of the requested data item. If an entry already exists, the S value of that entry is simply increased by $\frac{1}{s_i^2}$[1]. The server also updates the S-Heap, which is constructed based on the value of function $S(i)$. If no entry is found (i.e., there is currently no outstanding request for the data item), a new entry is created with the S value initialized as $\frac{1}{s_i^2}$ and $1stARV$ initialized to the current time. Then, the server inserts the new S value to S-Heap and adds the new $1stARV$ value to the end of D-Link, which is sorted based on $1stARV$ of each data item.

To make a broadcast scheduling decision, the server first checks the head of the D-Link to see whether the scheduling deadline has been passed. If the deadline has been passed, it deletes the head of the D-Link, removes the entry from the service queue, and removes the corresponding node from the S-Heap. Otherwise, it removes the root of the S-Heap, removes the corresponding entry from the service queue, and removes the corresponding node from the D-Link. The S-Heap or D-link operation (remove or insert) takes $O(\log N)$, where N is number of nodes in the heap (at most equal to the number of data items in the database). Thus, the scheduling decision overhead is $O(\log N)$. Compared to the decision overhead of LTSF, which is $O(N)$, our scheduling algorithm can significantly reduce the decision overhead.

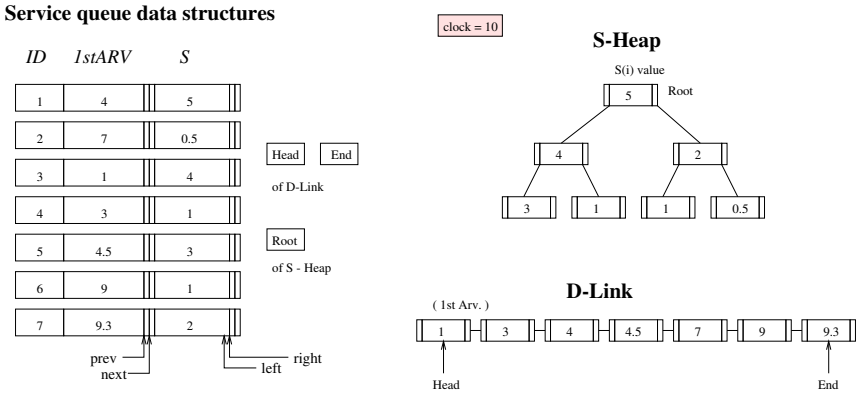

Figure 2: The service queue data structure

[1]To reduce the computation overhead, a variable can be used to represent $\frac{1}{s_i^2}$, and $\frac{1}{s_i^2}$ only needs to be calculated once.

4 Performance Evaluation

4.1 Simulation Model

We developed a simulation model written in CSIM [10]. The model represents an environment similar to that described in Section 2.1. Each client is simulated by a process which generates a stream of queries. Clients send requests to the server whenever the query is generated. The aggregated stream of requests, generated by the clients, follows a Poisson process with mean λ. The density function for the inter-arrival times of accessing item i is:

$$f(t) = \lambda_i e^{-\lambda_i t}$$

where $\lambda_i = p_i * \lambda$, p_i denotes the probability of clients requesting for data item i. The access pattern follows $Zipf$ distribution (similar assumptions are made by other researchers as well [4, 7, 17]). In the $Zipf$ distribution, the access probability of the i^{th} data item is represented as follows.

$$P_i = \frac{1}{i^\theta \sum_{j=1}^{n} \frac{1}{j^\theta}}$$

where $0 \leq \theta \leq 1$, n is the database size. When $\theta = 1$, it is the strict Zipf distribution. When $\theta = 0$, it becomes the uniform distribution. Large θ results in more "skewed" access distribution. The data item size varies from s_{min} to s_{max}, and has the following two types of distributions.

- **Random:** The distribution of data sizes falls randomly between s_{min} and s_{max}. Zipf distribution favors data items with small sequence numbers. Since the data size is random, the data size and the access distribution do not have correlation.
- **Increase:** The size (s_i) of the data item (i) grows linearly as i increases; i.e. $s_i = s_{min} + (i-1) * \frac{s_{max} - s_{min}}{n-1}$. In order words, the data item with smaller size will be accessed more frequently than bigger ones.

Most of the system parameters and their default values are listed in Table 2.

4.2 The Effects of the Access Pattern

Figure 3 shows the average stretch as a function of the access skew coefficient θ. As can be seen, our algorithm has the lowest average stretch. Since LTSF

Table 2: Default System Parameters

Database items	2000 items
Number of clients	200
Broadcast bandwidth (B)	115Kbps
The minimum data item size (S_{min})	1KB
The data item size ratio $R = \frac{S_{max}}{S_{min}}$	100
Mean query generate rate λ	$\frac{1}{100s}$
Zipf distribution parameter θ	0.6

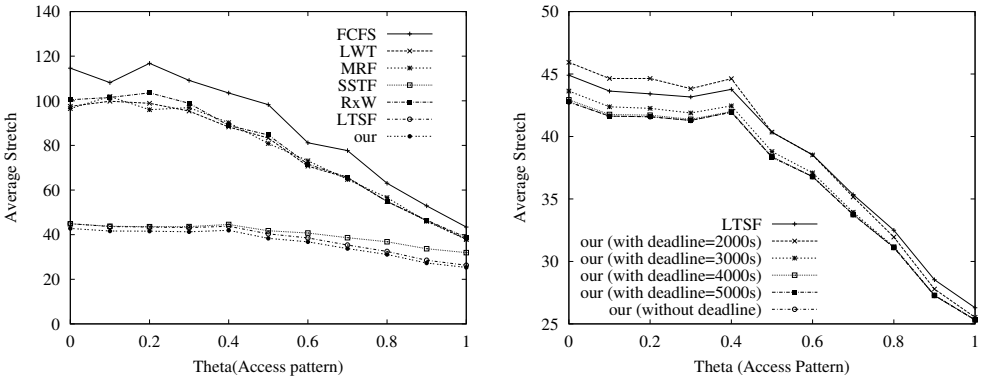

Figure 3: The average stretch under different access patterns (random distribution)

also considers stretch when making scheduling decisions, it has similar performance to ours. Since FCFS, LWT, MRF, and $R \times W$ do not consider stretch in broadcast scheduling, their average stretches are much higher than our algorithm and the LTSF algorithm. Note that SSTF has similar average stretch to our algorithm, since it broadcasts data in the order of data service time, which reflects the essence of the stretch-based algorithms.

Generally speaking, in order to reduce the average stretch, the server should try to schedule data items with small data size and high popularity. When $\theta = 0$, the access pattern is uniformly distributed, and data items have similar popularity. As a result, the data size is the major factor. Since FCFS, LWT, MRF, and $R \times W$ do not consider data size, they have much worse performance compared to our algorithm. The SSTF, however, considers the data size as major factor when making scheduling decisions, and it has similar average stretch to our algorithm when $\theta = 0$. When $\theta = 1$, the access pattern is much

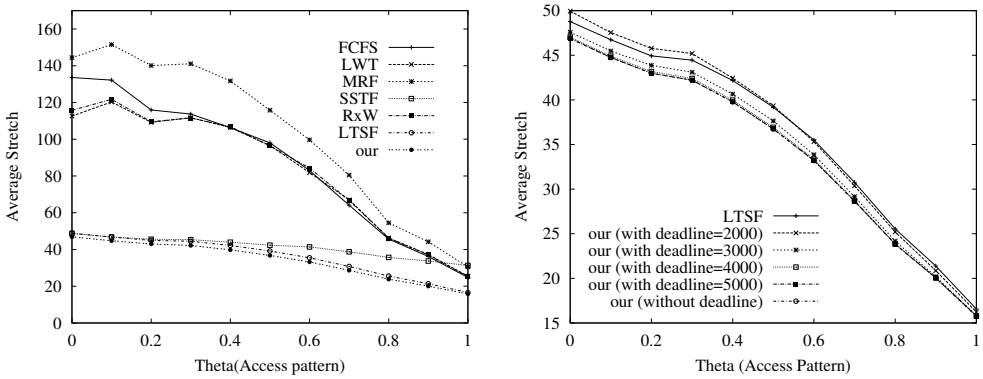

Figure 4: The average stretch under different access patterns (increase distribution)

more skewed, and the data popularity becomes a major performance factor. Since FCFS, LWT, MRF, and $R \times W$ consider the data popularity when making scheduling decisions, the performance difference between these algorithms and our algorithm becomes much smaller as $\theta = 1$. The SSTF algorithm, however, does not consider the popularity when making scheduling decisions, and its average stretch becomes much worse than our algorithm when $\theta = 1$.

The right graph of Figure 3 also shows the performance of our algorithm with different deadlines. It is easy to see that the average stretch reduces as the scheduling deadline increases. As expected, the worst-case waiting time decreases as the scheduling deadline increases (as shown in Figure 5). As shown in Figure 5, our algorithm has a comparative worst-case waiting time even without scheduling deadline. When the scheduling deadline is 2000, it has the best worst-case waiting time compared to other scheduling algorithms.

From Figure 3 and Figure 4, we can see that the difference between random distribution and the increase distribution is not that significant. Due to space limit, we only show results of the random distribution in the following sections.

4.3 The Effects of the System Work Load

Figure 6 shows the average stretch as a function of the mean query generate rate, the database size, and the number of clients. Three figures have similar trends; that is, as the work load (in terms of query generate rate, database size, or number of clients) increases, the average stretch also increases. From the figure, we can see that our algorithm outperforms other scheduling algorithms in various scenarios.

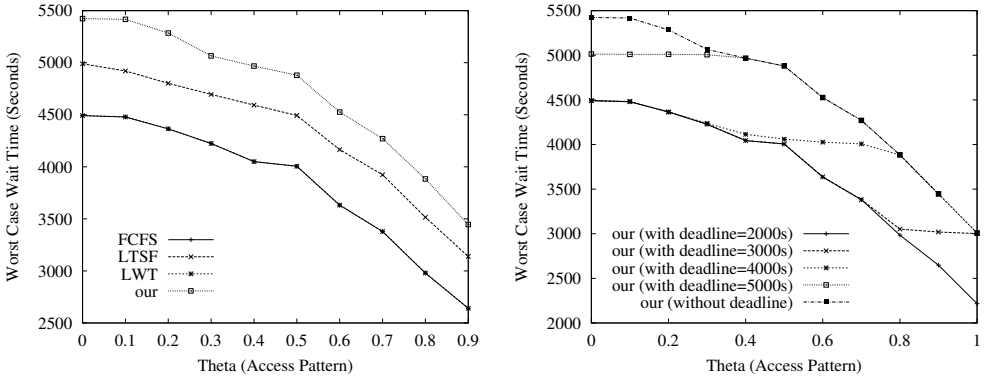

Figure 5: The worst waiting time under different access patterns

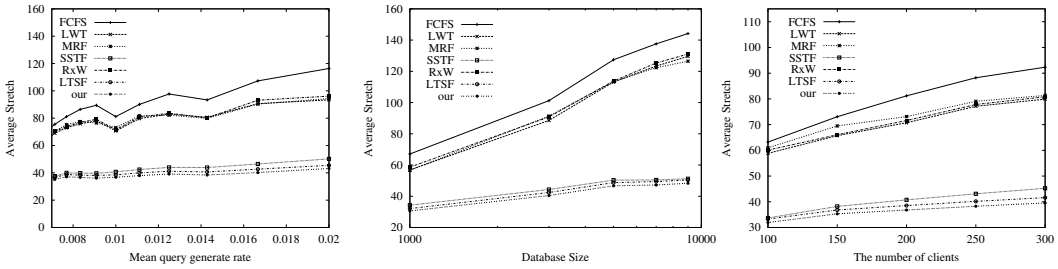

Figure 6: The average stretch under different system work load

4.4 The Scheduling Overhead

Figure 7 shows the scheduling overhead of the LTSF algorithm and our algorithm. In LTSF, the server has to recalculate the total stretch for every data item with pending requests in order to decide which data item to broadcast next, and hence the scheduling algorithm becomes a bottleneck due to its high computation overhead. As explained in Section 3.3, by using heaps, the computation complexity of our algorithm can be reduced to $O(\log N)$. Results from Figure 7 also verifies that our algorithm can significantly reduce the computation overhead compared to the LTSF algorithm. Moreover, in the LTSF algorithm, the computation overhead increases linearly when the database size or the number of clients increases, whereas the computation overhead in our algorithm increases much slower, and then our algorithm is more scalable compared to the LTSF algorithm.

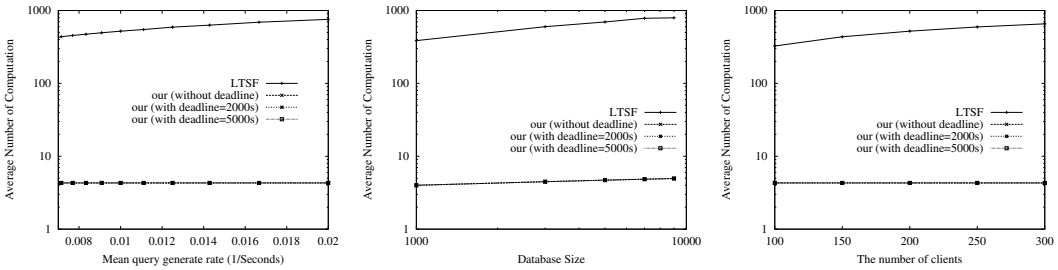

Figure 7: The scheduling overhead under different system work load.

5 Conclusions

This chapter has addressed the challenges of large-scale on-demand data broadcasts introduced by broadcast media such as satellite networks, cable networks, wireless LANs and cellular networks. In such environments, the scheduling problem is different from that in the point-to-point communication environment or the push-based broadcast environment. Moreover, when variable-sized heterogeneous requests are considered, most of the previous scheduling algorithm fail to perform well. As stretch is widely adopted as a performance metric for variable-size data requests, we proposed a broadcast scheduling algorithm to optimize the system performance in terms of stretch. One nice property of the proposed algorithm is that it is extremely simple and the computation overhead is very low. Analytical results described the intrinsic behavior of the algorithm. Simulation results demonstrated that our algorithm significantly outperforms existing scheduling algorithms under various scenarios.

References

[1] S. Acharya and S. Muthukrishnan. Scheduling On-Demand Broadcasts: New Metrics and Algorithms. *ACM MobiCom'98*, pages 43–54, Oct. 1998.

[2] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data Management for Asymmetric Communication Environments. *Proc. ACM SIGMOD*, pages 199–210, May 1995.

[3] D. Aksoy and M. Franklin. RxW: A Scheduling Approach for Large-Scale On-Demand Data Broadcast. *IEEE/ACM Transactions on Networking*, 7(6), Dec. 1999.

[4] M. Ammar and J. Wong. On the Optimality of Cyclic Transmission in Teletext Systems. *IEEE Transactions on Communications*, pages 8–87, Jan. 1987.

[5] A. Datta, D. Vandermeer, A. Celik, and V. Kumar. Broadcast Protocols to Support Efficient Retrieval from Databases by Mobile Users. *ACM Transactions on Database Systems*, 24(1):1–79, March 1999.

[6] H. Dykeman, M. H. Ammar, and J. Wong. Scheduling Algorithms for Videotext Systems under Broadcast Delivery. *In Proc. International Conference of Communications*, pages 1847–1851, 1996.

[7] S. Hameed and N. Vaidya. Efficient Algorithms for Scheduling Data Broadcast. *ACM/Baltzer Wireless Networks (WINET)*, pages 183–193, May 1999.

[8] T. Imielinski, S. Viswanathan, and B. Badrinath. Data on Air: Organization and Access. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):353–372, May/June 1997.

[9] R. Kalden, I. Meirick, and M. Meyer. Wireless Internet Access Based on GPRS. *IEEE Personal Communications*, pages 8–18, Apr. 2000.

[10] H. D. Schwetman. DSIM: A C-Based Process Oriented Simulation Language. *Proc. Winter Simulation Conf.*, pages 387–396, 1986.

[11] A. Silberschatz and P. Galvin. Operating System Concepts. *Addison Wesley*, 1997.

[12] C. Su and L. Tassiulas. Broadcast Scheduling for Information Distribution. *IEEE INFOCOM*, pages 107–117, 1997.

[13] Hughs Network Systems. DirecPC Home Page. *http://www.direcpc.com*, 2000.

[14] J.W. Wong. Broadcast Delivery. *Proceedings of the IEEE*, 76(12), Dec. 1999.

[15] X. Wu and V. Lee. Preemptive Maximum Stretch Optimization Scheduling for Wireless On-Demand Data Broadcast. *Proc. of Int'l Database Engineering and Applications Symposium (IDEAS)*, 2004.

[16] Y. Wu and G. Cao. Stretch-Optimal Scheduling for On-Demand Data Broadcasts. *IEEE International Conference on Computer Communications and Networks*, pages 500–504, Oct. 2001.

[17] L. Yin and G. Cao. Adaptive Power-Aware Prefetch in Wireless Networks. *IEEE Transactions on Wireless Communication*, 3(5):1648–1658, September 2004.