2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware

SLA-Aware Adaptive On-Demand Data Broadcasting in Wireless Environments

Adrian Daniel Popescu Mohamed A. Sharaf Cristiana Amza
Department of Electrical and Computer Engineering
University of Toronto
{adrian, msharaf, amza}@eecg.toronto.edu

***Abstract*—In mobile and wireless networks, data broadcasting for popular data items enables the efficient utilization of the limited wireless bandwidth. However, efficient data scheduling schemes are needed to fully exploit the benefits of data broadcasting. This motivated the proposal of several broadcast scheduling policies, which have mostly focused on either minimizing response time, or drop rate when requests are associated with hard deadlines. The inherent inaccuracy of hard deadlines in a dynamic mobile environment motivated us to use Service Level Agreements (SLAs) where a user specifies the utility of data as a function of its arrival time. Moreover, SLAs provide the mobile user with an already familiar quality of service specification from wired environments. Hence, in this paper, we propose SAAB which is an SLA-aware adaptive data broadcast scheduling policy for maximizing the system utility under SLA-based performance measures. To achieve this goal, SAAB considers both the characteristics of disseminated data objects as well as the SLAs associated with them. Additionally, SAAB automatically adjusts to the system workload conditions which enables it to constantly outperform existing on-demand broadcast scheduling policies.**

I. Introduction

With the continuous dependence on wireless data access, it is only natural that mobile users would expect the same *Service Level Agreements (SLAs)* currently provided to counterpart applications running on their stationary platforms in a wired environment. SLAs provide users with the flexibility to define the *utility* of delayed data. It also provides a concrete measure for system performance based on the users' perceived overall utility [14]. In this paper, we argue that SLA is a natural fit for the timely demands of mobile applications accessing time-critical data, including traffic and weather conditions, news headlines, stock quotes, and RSS feeds. Providing SLAs to such applications, together with mechanisms to enforce those SLAs, ensures the usability of mobile applications since they would provide end users with useful information within their pre-specified tolerated delays.

Enforcing SLAs requires re-thinking the design of current data dissemination techniques used in wireless networks, especially *data broadcasting*. Examples of data broadcasting systems include the Multimedia Broadcast Multicast Services (MBMS) [18] and the Terestrial digital broadcasting (ISDB-T) [17]. Towards exploiting these broadcasting capabilities, several data broadcasting schemes have been developed to reduce the delays in wireless data retrieval [1], [3], [4], [5]. These schemes could be broadly classified into push-based [1] and on-demand (i.e., pull-based) [3], [4], [5], where on-demand data broadcasting has been shown to be more scalable [6]. In particular, under on-demand data broadcasting users submit requests for data items of interest and the broadcast server aggregates requests for the same data item and broadcasts it only once. If a data item is highly popular, then broadcasting that data item to all interested users substantially reduces the number of transmissions. Hence, broadcasting allows for a more efficient utilization of the limited wireless bandwidth, resulting in shorter delays.

In order to fully reap the benefits of on-demand data broadcasting, an efficient request scheduler is needed so that to decide the best dissemination order of requested data with the goal of optimizing a certain performance metric. In this paper, we propose such a scheduler for optimizing performance under SLA. This is particularly important in mobile environments where data is accessed under certain time constraints. Examples of such constraints include the battery lifetime of the mobile device [8]. It also includes the spatiotemporal nature of the accessed data as in Location-based services where data is valid only within a local area [22]. Additionally, the SLA requirement is often imposed by the data provider itself so that to maintain a certain performance target.

Previous work on data broadcasting has focused mainly on optimizing the user perceived response time incurred in data retrieval [3], [4], [5], [19]. However, optimizing response time is not sufficient to maximize data usability since it overlooks the user's requirements and expectations. For instance, a user posing a request for available restaurants off an upcoming highway exit would have more stringent timely requirements than a user posing a request for the evening movie schedules.

As an alternative to response time, recent research on wireless data broadcasting has also looked at optimizing *drop rate* (also known as miss rate) [22], [8], [15]. Under such measure, each request is assigned a *hard deadline* and the system strives to minimize the number of requests missing their deadlines. However, one drawback of this approach is the complexity and inaccuracy entailed in setting those hard deadlines. Another problem with that approach is that it assumes data to be useless past the pre-specified hard deadline, hence it drops those requests which are prone to missing their deadlines.

With response time and drop rate being two extremes on the performance spectrum, SLAs have been used to successfully capture the users' performance expectations. Moreover, in a mobile environment, SLAs are inherently based on *soft dead-*

978-0-7695-3650-7/09 $25.00 © 2009 IEEE
DOI 10.1109/MDM.2009.25

lines, rather than hard deadlines, due to the following reasons. First, the user is likely to set deadlines based on *conservative time estimates* e.g., for reaching a particular target, such as an exit on a highway, or based on remaining battery lifetime. Second, if the deadline is not met, the requested update is likely still useful, and should be delivered; however, in this case, the service provider may be expected to pay a penalty, due to the potential inconvenience of the delay for the user.

In the simplest form of SLA, the utility of data is maximum if it arrives before the soft deadline, then that utility decreases linearly with time. This simple form of SLA is known as *tardiness*. In the more general form, the utility could be expressed using any monotonic function rather than restricting it to the linear one.

Towards maximizing the utility of on-demand data broadcasting, we propose an SLA-Aware Adaptive Broadcast scheduling policy called SAAB. In particular, we first propose SAAB for optimizing the simple form of SLA defined in terms of tardiness. Further, we extend SAAB to optimize the general form of SLA defined by monotonic utility functions.

In both versions of SAAB, one important challenge is the impact of request aggregation on the scheduling problem. Request aggregation is efficient since it allows for fewer data broadcasts which saves bandwidth and reduces delays. However, in the presence of SLAs, different requests for the same data item might have different SLA specification which are often in conflict. SAAB exploits such variability in SLAs in order to maximize the overall user satisfaction as defined in terms of SLAs.

Moreover, our proposed SAAB policy is adaptive to workload conditions which allows it to consistently provide the best achievable performance. In contrast, we show that existing schedulers for optimizing response time and drop rate are sometimes also successful in optimizing performance under SLAs for some workload conditions but not all. In fact, such schedulers might perform the best under a certain workload but perform the worst under another workload.

The rest of the paper is organized as follows. Section II introduces the system model. Section III presents the tardiness metric and our tardiness-aware scheduling policy. Section IV extends tardiness to its more general form of utility function and also presents our extended utility-aware scheduling policy. Section V follows with the experimental setup. Section VI presents our experiments. Related work is presented in Section VII. Finally, Section VIII concludes our paper.

II. System Model

We assume a typical on-demand data broadcasting environment where there is a single data server providing information to multiple mobile clients. The communication between the server and clients is established through both an uplink and a downlink data channels. Specifically, clients send requests for data items on the uplink channel, whereas the server disseminates the corresponding data items on the downlink channel.

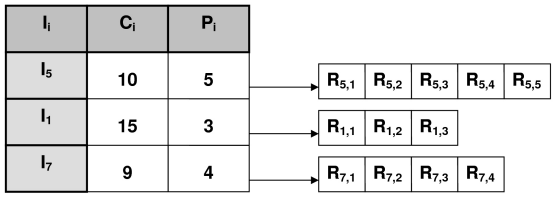

Fig. 1. Example of the Scheduling Queue

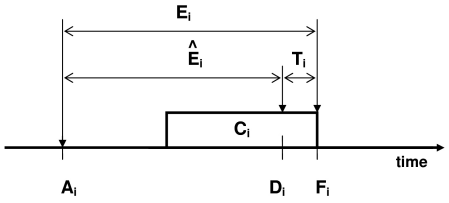

Fig. 2. Notations associated with request R_i

The server maintains a database of N data items $< I_1, I_2, ...I_N >$ where each data item I_x has a unique size L_x. For data transmission, each data item is segmented and broadcast as a set of equal-sized packets, where a packet is the smallest logical unit of data transmission.

A request R_i issued by client for a certain data item is characterized by the following parameters:

1) Data Item (I_i): which is the data item corresponding to request R_i,
2) Arrival Time (A_i): which is the point of time where request R_i is issued, and
3) Deadline (D_i): which is the soft deadline associated with request R_i.

There are multiple alternatives for setting the soft deadline D_i which could be generally categorized as either static or dynamic. In the static case, it is set according to some quality contract established between the user and the service provider. For instance, it might state that a stock quote retrieval request should be satisfied within one second. In the dynamic case, it is set according to the user's current need for data. For instance, the time when the user is expected to hit a highway exit would act as a deadline for the exit-related information. Under utility-based SLAs, these soft deadlines are also associated with some utility function. For the sake of simplicity, we will defer that discussion to Section IV.

As data requests arrive at the server, they are queued internally in a scheduling queue. In the scheduling queue, multiple requests for the same data item I_j are aggregated together. In particular, each entry in the scheduling queue corresponds to a single data item in the database, where each entry is represented as follows:

1) Service Time (C_j): is the time required for transmitting data item I_j on the downlink channel,

2) Popularity (P_j): is the number of pending requests for data item I_j, and
3) Requests Vector ($\mathbb{R}_j$): is a requests vector of length P_j, where each entry in $\mathbb{R}_j$ corresponds to one of the P_j pending requests for I_j (i.e., $\mathbb{R}_j = R_{j,1}, R_{j,2}, ..., R_{j,P_j}$.

Figure 1 shows a sample scheduling queue which contains 3 data items with pending requests. For instance, the transmission cost of data item I_5 is 10 time units and it has 5 pending requests $< R_{5,1}, ..., R_{5,5} >$.

The time when request R_i is satisfied is denoted as *finish time* (F_i). That is the time when the user finishes downloading the data item I_i requested in R_i. Ideally, F_i should be less than or equal to D_i. Equivalently, the ideal response time for request R_i is $\hat{E}_i = D_i - A_i$. However, R_i might experience queuing delays in the server leading to delaying the finish time of R_i beyond D_i resulting in a perceived response time $E_i = F_i - A_i$ (as shown in Figure 2).

Broadcast data servers typically employ a request scheduler in order to optimize a certain performance metric such as the perceived response time described above. In the next section, we present examples of such schedulers.

A. Scheduling Policies in Broadcast Systems

In a broadcast data server, a request scheduler decides the transmission order of data items according to some prioritizing policy, where each data item is assigned a priority according to certain criteria. Hence, whenever more than one data item are being requested, the data item with the highest priority is served first.

Several request scheduling policies have been proposed in the literature. These policies basically extend or integrate the following three basic policies: 1) Shortest Job First (SJF), 2) Earliest Deadline First (EDF), and 3) Most Requests First (MRF). Formally, we define the priority functions for these three policies as follows:

1) SJF: Each data item I_i is assigned a priority equal to $\frac{1}{C_i}$, where C_i is the transmission cost of I_i.
2) EDF: Each data item I_i is assigned a priority equal to $\frac{1}{D_i}$, where D_i is the minimum deadline among all the requests for I_i.
3) MRF: Each data item I_i is assigned a priority equal to P_i, where P_i is the number of pending requests for I_i.

Previous work has extended the above basic policies either to: 1) optimize response time, or 2) minimize drop rate. For instance, towards optimizing response time, [4] proposes the RxW policy which is a combination of MRF and First-Come-First-Served. Similarly, a policy which considers both popularity and transmission cost has appeared in [16], [7]. In this paper, we will refer to such policy as Weighted Shortest Job First (W-SJF) where the priority function under W-SJF is computed as $\frac{P_i}{C_i}$.

Similarly, several policies have been proposed with the objective of minimizing drop rate (i.e., the number of requests not meeting their hard deadlines). Examples of such policies include MAI [8] which basically assigns each data item a priority value equal to the product of all the three priority functions listed above. Similar proposals have also appeared in [22], [15].

III. Tardiness-Aware Scheduling

In this section, we first present the tardiness-based performance metric, then we discuss the problem of optimizing data broadcasting under such metric and finally, we present our SAAB request scheduling policy for optimizing tardiness. Our SAAB policy for maximizing utility is presented in Section IV.

A. The Tardiness Metric

Typically, the broadcast server strives to satisfy each request R_i before its deadline. However, if R_i cannot meet its deadline, the system will still satisfy it but it will incur some *tardiness* T_i which accounts for the amount of deviation from the pre-specified SLA (i.e., deadline). Hence, $T_i = 0$ iff $F_i \leq D_i$, and $T_i = F_i - D_i$ otherwise.

In contrast to the scheduling policies presented in Section II, in this paper, we focus on soft deadline-based metrics. This includes the tardiness metric defined above and the more general utility metric defined later in Section IV.

Towards optimizing tardiness, it has been shown that EDF outperforms SJF at low system utilization, whereas at high utilization, SJF significantly outperforms EDF [20], [10]. This has been observed in the context of transaction scheduling where each transaction is submitted by a single user. However, in the data broadcast scheduling problem addressed in this paper, a data item is typically requested by several users.

This request aggregation is efficient since it allows for fewer data broadcasts which saves bandwidth and reduces delays. However, in the presence of SLAs, different requests for the same data item might have different deadlines which are often in conflict. SAAB exploits such variability in deadlines in order to maximize the overall performance as explained next.

B. Tardiness-Aware SAAB

In order to describe out SAAB scheduling policy, let's first assume an entry in the requests queue for data item I_x where I_x has a transmission cost C_x. Further, assume there is some pending request R_x for that data item I_x with deadline D_x. Finally, let S_x be the current *slack* of request R_x if it is served immediately. Hence, S_x of request R_x is computed as: $S_x = D_x - (t + C_x)$, where t is the current time. In the case where data item I_x has P_x pending requests, then we denote the slacks for these requests as $< S_{x,1}, S_{x,2}, ..., S_{x,P_x} >$ and they are computed similarly.

In order to decide if I_x should be scheduled for transmission, SAAB assigns I_x a priority value which is computed according to a certain priority function. To illustrate the priority function employed by SAAB, let us assume two data items I_1 and I_2 with transmission costs C_1 and C_2 respectively. Further, assume that the number of pending requests for item I_1 is P_1 and the number of pending requests for item I_2 is P_2. Hence, the slacks for I_1 is $< S_{1,1}, S_{1,2}, ..., S_{1,P_1} >$. Similarly, the slacks for I_2 is $< S_{2,1}, S_{2,2}, ..., S_{2,P_2} >$. Finally, assume a

schedule X where I_1 is scheduled for transmission first then I_2, and another alternative schedule Y where I_2 is scheduled first then I_1.

For X to be the schedule of choice, then the tardiness provided by X should be less than that of Y. Under schedule X, where I_1 is followed by I_2, the total tardiness of requests for data item I_1 is simply the sum of all requests which currently have a negative slack. These are the requests which are currently tardy by "default" at the time where the scheduling decision is made. Let us assume that the number of these requests is $P_{1,def}$, where $0 \leq P_{1,def} \leq P_1$. Hence, the tardiness of requests for item I_1 under schedule X is:

$$T_{1,X} = \sum_{j}^{P_{1,def}} (-S_{1,j})$$

When I_1 is scheduled first, then each request for I_2 is delayed by the amount of time needed to transmit I_1 (i.e., C_1). Hence, the slack for each those requests decreases by C_1 time units and the tardiness is affected as follows:

1) Increasing the tardiness of requests which were already tardy by default at the time where the scheduling decision was made (i.e., $P_{2,def}$),
2) Rendering additional tardy requests to I_2. These are the requests which had positive slack when the scheduling decision was made but that slack became negative during waiting for the transmission of I_1. Lets assume the number of those "additional" requests is $P_{2,add}$.

Accordingly, the total tardiness of requests for data item I_2 is simply the sum of all requests which will have a negative slack after transmitting I_1 (i.e., $P_{2,def} + P_{2,add}$) where $0 \leq P_{2,def} + P_{2,add} \leq P_2$. Hence, the tardiness of requests for item I_2 under schedule X is:

$$T_{2,X} = \sum_{j}^{P_{2,def}+P_{2,add}} (C_1 - S_{2,j})$$

Accordingly, the total tardiness (T_X) under schedule X is computed as follows:

$$T_X = \sum_{j}^{P_{1,def}} (-S_{1,j}) + \sum_{j}^{P_{2,def}+P_{2,add}} (C_1 - S_{2,j})$$

In a similar manner, under schedule Y, where item I_2 is broadcasted before item I_1, the total tardiness is computed as follows:

$$T_Y = \sum_{j}^{P_{2,def}} (-S_{2,j}) + \sum_{j}^{P_{1,def}+P_{1,add}} (C_2 - S_{1,j})$$

In order that T_X to be less than T_Y, and after applying several computations we obtain that:

$$\frac{1}{C_1}(P_{1,def} + P_{1,add} + \sum_{j}^{P_{1,add}} -\frac{S_{1,j}}{C_2})$$

has to be greater than:

$$\frac{1}{C_2}(P_{2,def} + P_{2,add} + \sum_{j}^{P_{2,add}} -\frac{S_{2,j}}{C_1})$$

The above inequality allows us to compare the priorities of items I_1 and I_2. In general, in order to compare item I_i with any arbitrary item from the scheduling queue, we substitute C_2 with $\tilde{C}$, where $\tilde{C}$ is average transmission cost of any data item. Hence, we obtain the priority:

$$Pr_i = \frac{1}{C_i}(P_{i,def} + P_{i,add} + \sum_{j}^{P_{i,add}} -\frac{S_{i,j}}{\tilde{C}})$$

where $P_{i,add}$ is the number of requests for item I_i where $S_{i,j} - \tilde{C} < 0$.

In the above priority function, $\tilde{C}$ represents the delay incurred by I_i if another single arbitrary data item is scheduled first. However, in the general case, we want to compare I_i against all the data items with pending requests. Hence, our general priority function is defined as:

$$Pr_i = \frac{1}{C_i}(P_{i,def} + P_{i,add} + \sum_{j}^{P_{i,add}} -\frac{S_{i,j}}{(n-1)\tilde{C}}) \quad (1)$$

where n is the number of data items with pending requests, $(n-1)\tilde{C}$ is the total delay incurred by requests for I_i if the other $n-1$ data items from the queue are scheduled first, and $P_{i,add}$ is the number of requests for item I_i where $S_{i,j} - (n-1)\tilde{C} < 0$ (i.e., $\frac{S_{i,j}}{(n-1)\tilde{C}} < 1$).

IV. Utility-Aware Scheduling

In this section, we first present the utility-based performance metric, then we discuss the relationship between tardiness and utility as two forms of SLA and finally, we present our SAAB request scheduling policy for maximizing utility.

A. The Utility Metric

This general form of SLA allows users to define the *utility* of requested data as a function in response time. Typically such utility function consists of two components: The first component specifies the maximum utility when data is received before its deadline and the second component specifies a monotonically decreasing function which defines the utility of data received after its deadline.

For instance, consider the utility function described in [14] and illustrated in Figure 3. For that utility function, the user expects the response time E_i for request R_i to be less than or equal to $\hat{E}_i$, where $\hat{E}_i = D_i - A_i$. Under such function, the utility gained by the system when request R_i is satisfied is defined as:

$$u_i(E_i) = \begin{cases} V_i & if E_i \leq \hat{E}_i \\ V_i e^{\alpha_i(E_i - \hat{E}_i)} & if E_i > \hat{E}_i \end{cases} \quad (2)$$

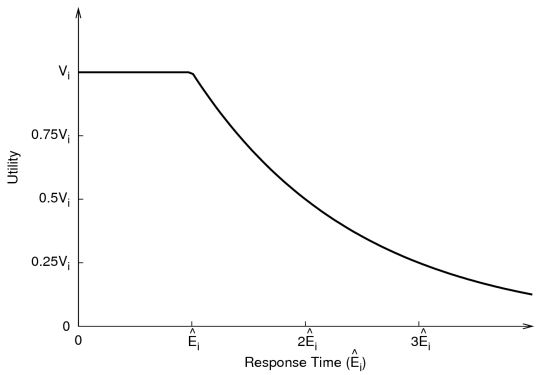

Fig. 3. Sample utility function

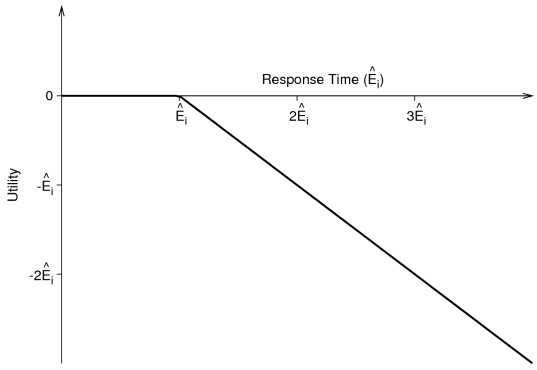

Fig. 4. Tardiness expressed as utility function

Under such utility function, the system gains the maximum value (i.e., V_i) as long as R_i is satisfied before its deadline D_i (or equivalently, if $E_i \leq \hat{E}_i$). This gain decreases exponentially as E_i exceeds $\hat{E}_i$. For instance, with $\alpha_i = \frac{ln0.5}{\hat{E}_i}$, the utility for a request R_i decays by half for every order of increase in the response time.

B. Tardiness vs. Utility

The tardiness form of SLA presented in Section III represents a special case of the general utility form of SLA. In particular, under the tardiness metric, the broadcast system is not penalized as long as a request is satisfied before its deadline. However, if the data item is received after its deadline, the system is penalized where that penalty increases linearly with delay. For instance, consider a request R_i for data item I_i with arrival time A_i and deadline D_i. Hence, the expected response time for that request is $\hat{E}_i = D_i - A_i$. The utility for such request is illustrated in Figure 4 and formulated as:

$$u_i(E_i) = \begin{cases} 0 & if E_i \leq \hat{E}_i \\ -(E_i - \hat{E}_i) & if E_i > \hat{E}_i \end{cases} \quad (3)$$

As shown in Figure 4, accumulating tardiness resembles a negative utility value. In particular, the broadcast system gains maximum utility (=0) if the actual response time E_i of R_i is within the timeframe expected by the user (i.e., $E_i \leq \hat{E}_i$). However, the utility gained by the system decreases as E_i increases (i.e., the system is penalized). For instance, the system gains a utility value of $-\hat{E}_i$ if the request response time is $2\hat{E}_i$. Equivalently, the system is penalized $\hat{E}_i$ if the request response time is $2\hat{E}_i$.

Establishing this relationship between tardiness and utility enables us to extend our SAAB scheduling policy for the goal of optimizing utility as explained in the next Section.

C. Utility-Aware SAAB

In order to schedule for optimizing utility, we use the same technique as in Section III. In particular, let's assume two data items I_1 and I_2 with transmission costs C_1 and C_2 respectively. Further, let's assume that data item I_1 has P_1 pending requests (i.e., $< R_{1,1}, R_{1,2}, ..., R_{1,P_1} >$), where $E_{1,j}$ is the response time of request $R_{1,j}$ if I_1 is scheduled for broadcasting right now. Similarly, I_2 has P_2 pending requests (i.e., $< R_{2,1}, R_{2,2}, ..., R_{2,P_2} >$), where $E_{2,j}$ is the response time of request $R_{2,j}$ if I_2 is scheduled for broadcasting right now.

In a scheduler X, where item I_1 is broadcasted before item I_2, the response time of request $R_{1,j}$ is $E_{1,j}$ since I_1 is scheduled immediately. However, each request for I_2 is delayed by the amount of time needed to transmit I_1 (i.e., C_1). Hence, the response time for each request for I_2 increases by C_1 time units. Accordingly, the total utility U_X under schedule X is computed as:

$$U_X = \sum_{j}^{P_1} u_{1,j}(E_{1,j}) + \sum_{j}^{P_2} u_{2,j}(C_1 + E_{2,j})$$

Similarly, in a scheduler Y, where item I_2 is broadcasted before item I_1, the total utility U_Y under schedule Y is computed as:

$$U_Y = \sum_{j}^{P_2} u_{2,j}(E_{2,j}) + \sum_{j}^{P_1} u_{1,j}(C_2 + E_{1,j})$$

To achieve the general priority function, we extend the comparison to the general case where we compare an arbitrary item I_i with the other items in the scheduling queue as explained earlier in Section III. Accordingly, the general priority function for optimizing tardiness is defined as:

$$Pr_i = \sum_{j}^{P_i} u_{i,j}(E_{i,j}) - \sum_{j}^{P_i} u_{i,j}(E_{i,j} + (n-1)\tilde{C}) \quad (4)$$

where n is the number of data items with pending requests , $(n-1)\tilde{C}$ is the total increase in response time incurred by each request for I_i if the other $n-1$ data items in the queue are scheduled first.

To understand the intuition underlying our SAAB policy for optimizing utility, notice that the priority function above (Eq. 4) computes the *loss* in utility incurred by I_i if all other

data items in the queue are scheduled first. In particular, SAAB employs a *greedy* heuristic where it gives the highest priority to the data item which is penalized the most if it is scheduled last.

V. Experimental Setup

We have created a simulator to model an on-demand data broadcast server where mobile clients submit requests for data items stored at the server. In the following, we present the settings that we used in generating the workload, as well as the scheduling policies taken into consideration. Experimental results are presented in the next section.

Data items: We model a database of 10,000 data items where the size of each item is picked from the range [s_{min}–s_{max}] according to a Zipf distribution. The default size range is [50K–150K] and the skewness parameter (θ_{size}) for the Zipf distribution is 1.0 resulting in more small-size items in the database. Additionally, the popularity of each item is set according to a Zipf distribution with a skewness parameter (θ_{pop}) of 0.5. By default, if not specified otherwise, there is no correlation between the popularity of a data item and its size.

Deadlines: As already specified in the model section, Each request is associated with a soft deadline where the deadline D_i for request R_i is set as: $D_i = A_i + (1 + SF_i) \times C_i$, where A_i is the arrival time of R_i, C_i is the time (cost) for transmitting the data item requested by R_i, and SF_i is the slack factor parameter which determines the ratio between the initial slack time of a request and the data transmission cost. SF_i is a number uniformly distributed in the interval [0, SF_{max}], where SF_{max}, the maximum slack factor, is a simulation parameter.

Requests: The inter-arrival times of requests follow an exponential distribution with the mean inter-arrival time being set to achieve utilizations in the range [0.0–1.0]. For the sake of completeness, in our experiments we consider two request generation settings where each element in the generated exponential sequence represents either: 1) a *batch* of requests, or 2) an *isolated* request. In the case of batches, all requests in a batch are for the same data item and they all have the same arrival time but different deadlines, whereas for the isolated case only one request is generated. The batch setting emphasizes request aggregation, hence it is especially useful for studying the interplay between high popularity with discrepancy in deadlines. To the contrary, the isolated setting still allows us to study system performance under lower degrees of request aggregation. In the next section, we present experimental results under both the batch and isolated settings with the batch setting being the default.

Utility: Each request R_i is also associated with a utility function u_i as specified in Section IV. Clearly, some data items are by nature more important than others (e.g., stock quotes or traffic conditions). Additionally, that importance vary from one user to another depending on the situation. In order to capture that dual nature of utility, we assign classes to both data items (K_D) and requests (K_R). In particular, we divided the items in the database into 3 classes, where the first third of data items belong to class $K_D = 0$, and the following 2 thirds belong to classes $K_D = 1$ and $K_D = 2$ respectively. Moreover, in order to highlight the trade-off between utility and popularity, we set the data utility classes so that utility is inversely correlated with popularity. Further, each request R_i for a data item I_i is assigned a utility function where the maximum utility of the request (i.e., V_i) is set to the value 10^{K_R} where K_R is generated according to a Zipf distribution in the range [0–2]. In particular, K_R is generated with skewness parameter ($\theta_{utility}$) which is highly skewed towards the utility class K_D assigned to the requested data item. Specifically, requests for items in the class $K_D = 0$ have mostly V_i=1, whereas requests for items in $K_D = 1$ and $K_D = 2$ have mostly V_i=10 and mostly V_i=100 respectively.

Policies: We compare the performance of Earliest Deadline First (EDF) and Weighted Shortest Job First (W-SJF) against SAAB-T and SAAB-U, our two proposed versions of the SAAB policy for optimizing tardiness and utility respectively. Additionally, we also compare the performance of SAAB against the following two policies:

1) Slack Inverse Number of requests (SIN) [22]
2) Multi-Attribute Integration (MAI) [8]

Under SIN, the priority assigned to a data item I_i is set as: $Pr_i = (D_i - t)/P_i$, where t is the current time and D_i is the earliest deadline among all the pending requests for I_i.

Notice that under SIN, the data item with *lowest* Pr_i value is scheduled first. Additionally, for SIN, the higher the popularity (i.e., P_i) of I_i, the lower the value Pr_i. However, this only holds as long as $D_i - t \geq 0$. In the cases where $D_i - t < 0$, that relationship is reversed. However, under SIN it is always true that $D_i - t \geq 0$. This because SIN deals with hard deadlines where a request is dropped if it misses its hard deadline.

Since in this paper, we consider soft deadlines rather than hard deadlines, we have modified SIN so that it can handle soft deadlines while maintaining the same logic of the original SIN. In particular, we set the priority under SIN as follows: $Pr_i = (D_i - t)/P_i$, when $D_i - t \geq 0$ and $Pr_i = (D_i - t) \times P_i$, when $D_i - t < 0$.

Similarly, we have modified the priority function for MAI as follows: $Pr_i = (C_i \times S_i)/P_i$, when $S_i \geq 0$ and $Pr_i = (P_i \times S_i)/C_i$, when $S_i < 0$.

Our experiments show that under soft deadlines, our modified versions of SIN and MAI provide better results than the original versions. Hence, throughout the rest of this paper, we will only consider the modified SIN and MAI in our comparisons.

VI. Experiments

We have conducted several experiments to evaluate the performance of our proposed *SAAB* policy. In this section, we present a representative sample of our results. All the results reported here are the average of 10 simulation runs.

Tardiness - Default Settings: In our first experiment, we set the skewness in items popularity θ_{pop} to 0.5, the skewness

TABLE I
WORKLOAD PARAMETERS

Parameter	*Symbol*	*Default Setting*	*Other Settings*
Number of data items	N	10,000	100
Skewness in popularity	θ_{pop}	0.5	
Minimum data item size	s_{min}	50 KB	
Maximum data item size	s_{max}	150 KB	250 KB
Item size granularity	sg	1 KB	
Skewness in size distribution	θ_{size}	1	2
Number of clients	NC	10,000	
Actual slack factor	SF	[0, SF_{max}]	
Maximum slack factor	SF_{max}	12	4, 6, 18
Skewness in utility distribution	$\theta_{utility}$	2	

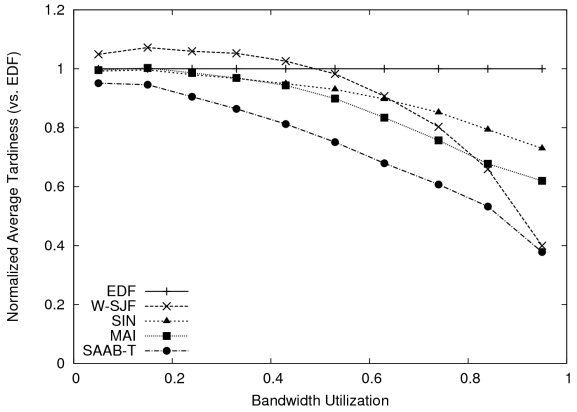

Fig. 5. Tardiness for Default Settings ($SF_{max} = 12$)

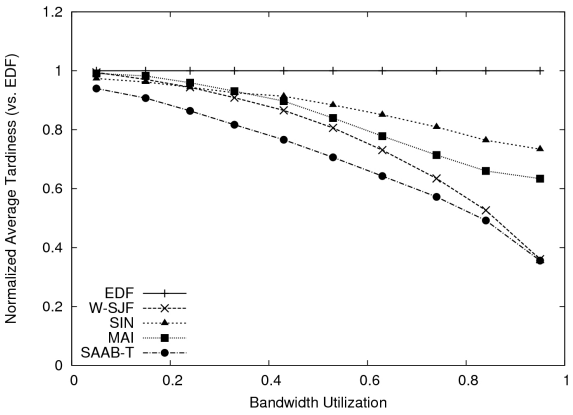

Fig. 6. Impact of Slack Factor on Tardiness (SF_{max}=6)

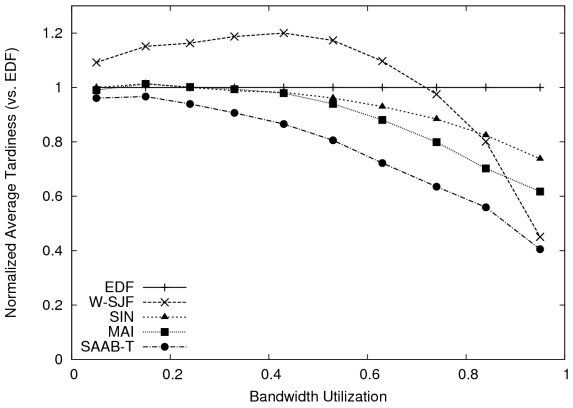

Fig. 7. Impact of Slack Factor on Tardiness (SF_{max}=18)

in item size distribution θ_{size} to 1 and the maximum slack factor SF_{max} to 12, as provided by the default settings (see Table I). Figure 5 shows the average tardiness for each policy normalized to EDF for utilization points between 0.0 and 1.0.

The figure shows that EDF, MAI, and SIN outperform W-SJF only at low utilizations, whereas W-SJF is performing better at high utilizations. The reason is that at low utilization, the system is able to meet most of the deadlines, and hence, under deadline-aware policies most requests are satisfied before their deadlines without experiencing any tardiness. As the utilization grows, the system cannot meet as much deadlines, and hence the deadline-aware policies enter the domino effect phase where requests keep missing deadlines and accumulating tardiness.

On the other hand, SAAB, being adaptive to the workload, it consistently outperforms the other policies for all values of utilization. However, the maximum improvement achieved by SAAB is at medium utilization around 0.5 (notice that this the same point where W-SJF starts to outperform EDF). At that point, SAAB reduces tardiness by 25% compared to EDF, by 23% compared to W-SJF and about 16% when compared to MAI.

As the utilization increases beyond 0.5, we observe that policies which take item size into consideration (i.e., SAAB, W-SJF and MAI) perform significantly better than EDF and SIN. Moreover, as we approach the utilization of 1.0, the performance of SAAB is very similar to that of W-SJF. This is due to the fact that the priority function used by SAAB is similar to that of W-SJF at high utilization where most requests miss their deadlines.

Tardiness - Impact of Slack Factor: In this experiment we measure the impact of varying the maximum slack factor SF_{max} on the average tardiness. Figure 6 shows the results obtained for a maximum slack factor of 6. In this case the deadlines are much tighter than in the previous experiment where $SF_{max} = 12$. Hence, W-SJF performs better than EDF even at low utilizations. The reason is that EDF enters into the domino effect faster when deadlines are tighter.

We can also observe that MAI and SIN perform much better than EDF, as they consider in addition to the deadline, item popularity and/or the item size. However, the tardiness provided by these 2 approaches is still higher than both W-SJF and SAAB.

Figure 7 shows the results obtained for a maximum slack factor of 18. Here, as in the previous experiment, W-SJF performs worse than EDF at low and moderate utilizations. The reason is that the slack is big enough to allow EDF to perform better than W-SJF until the utilization of 0.7. At this point SAAB-T provides an improvement of 20% when

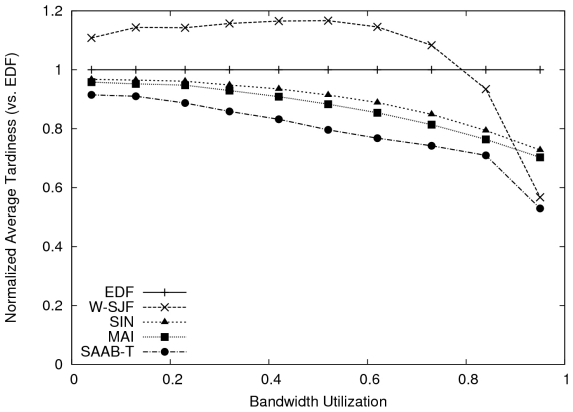

Fig. 8. Impact of Skewness in Size Distribution on Tardiness (θ_{size}=2)

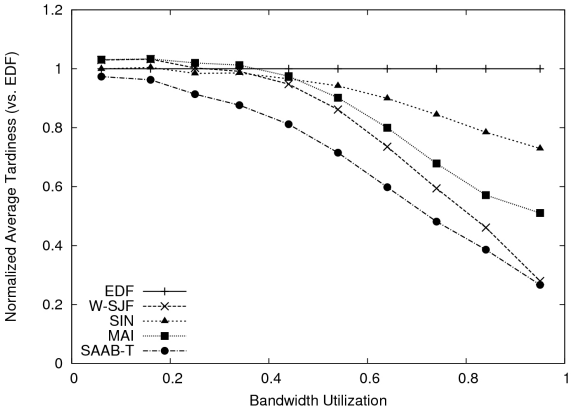

Fig. 9. Impact of Item Size Bounds on Tardiness ($s_{max} = 250K$)

compared to MAI, 27% when compared to EDF, and an improvement of almost 34% when compared to W-SJF.

Tardiness - Impact of Skewness in Item Size Distribution: In this experiment, we study the impact of the skewness parameter of item size distribution θ_{size}. In particular, we further study the performance at $\theta_{size} = 2.0$ (Figure 8) in addition to the default setting of $\theta_{size} = 1.0$ (Figure 5).

Comparing Figures 8 and 5, we notice that for $\theta_{size} = 2.0$, EDF performs better than W-SJF for a longer time (i.e., higher utilization). This is because in this experiment, the smaller-sized items are more popular than in the default setting (Figure 5), thus there is a higher probability that the item which has the earliest deadline will also have a small size. SAAB still outperforms the other policies due to its adaptivity characteristics.

Tardiness - Impact of Item Size Bounds: In this experiment, we change the item size bounds to see its impact on the average tardiness. Figure 9 shows the results for s_{min}=50K and s_{max}=250K. Hence, under that setting the ratio between the largest and smallest data item is 5.

For this case, we can see a clearer distinction between the performance provided by the different policies. Specifically, the policies which take item size into consideration (i.e., SAAB, W-SJF, and MAI) perform much better at the average-high utilization than the policies which overlook that parameter (i.e., EDF and SIN). The reason is that when the ratio in size between the smallest and largest item increases, broadcasting a large-sized item will have a more prominent impact in delaying the small-size items in the scheduling queue. With SAAB being adaptive to workload condition including variability in data item size, it consistently outperforms all the other policies.

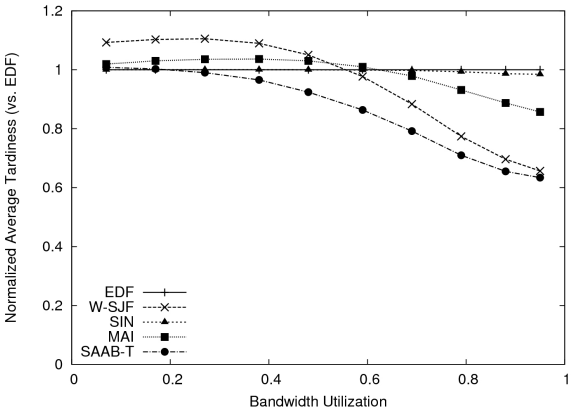

Fig. 10. Tardiness for the Isolated Request Generation Setting

Tardiness - Isolated Request Generation Setting: For the sake of completeness, in this experiment we generate requests according to the isolated request generation setting described in Section V. In order to still allow for request aggregation under such setting, we use a database with only 100 data items. Further, we set s_{min}=50K, s_{max}=250K and the slack factor SF_{max}=4. For the rest of the parameters we use the default settings as specified in Table I. In Figure 10 we observe that SAAB-T achieves up to 13% improvement as compared with W-SJF, and up to 37% improved as compared with EDF. Similar to the batch setting, EDF based policies (EDF, SIN, MAI) perform better than W-SJF only at small-medium utilization points. Around utilization point 0.6 W-SJF starts to outperform the other policies and it approaches SAAB-T at the high end of utilization.

Utility - Default Settings: In this experiment we study the utility performance of our policy SAAB-U where we set the utility functions as described in Section V. Figure 11 shows the total utility provided by each policy normalized to the ideal total utility which is only achieved if all requests are satisfied before their deadlines. Hence, the higher the value, the closer the scheduling policy to the ideal performance. The figure shows that at high utilization, SAAB-U improves the utility by up to 12% compared to W-SJF and by up to 58% compared to EDF. These improvements are expected since none of the other scheduling policies takes utility into consideration.

Utility - Impact of Correlation between Popularity and Size Distributions: Previous work on data broadcast systems has studied two special cases for the correlation between the popularity of a data item and its size, namely, *Decrement (DEC)* and *Increment (INC)* [11], [15]. Similarly, in this paper,

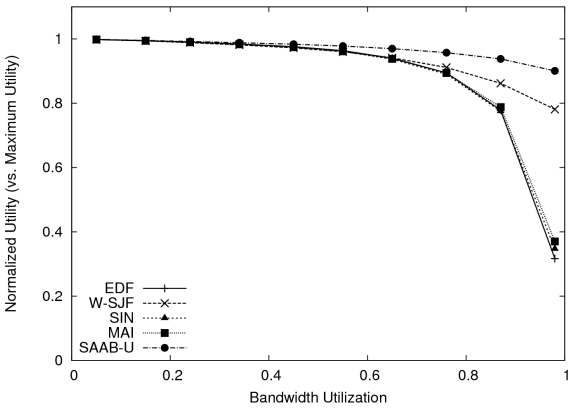

Fig. 11. Utility for Default Settings

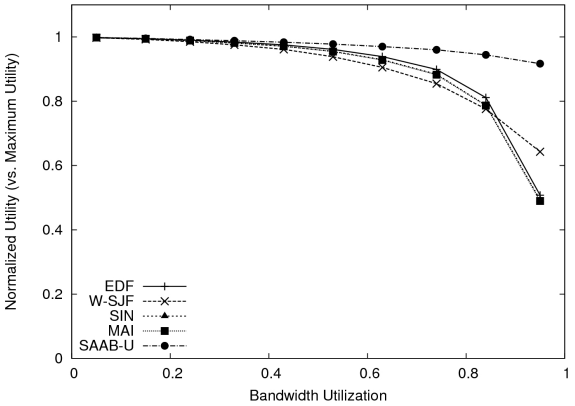

Fig. 12. Impact of DEC Correlation on Utility

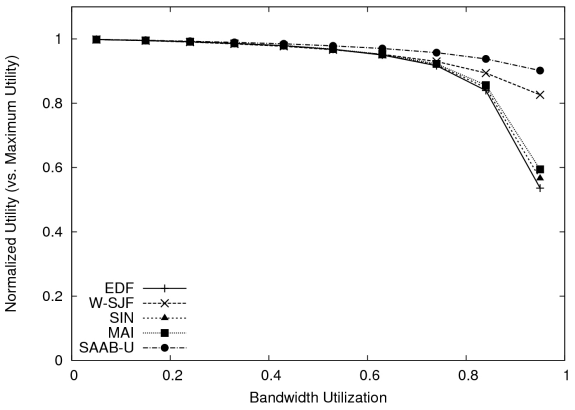

Fig. 13. Impact of INC Correlation on Utility

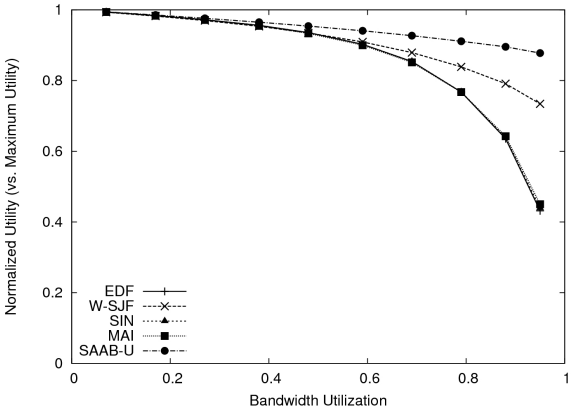

Fig. 14. Utility for the Isolated Request Generation Setting

we also study the performance under those two settings when maximizing utility is the desired performance goal.

Figure 12 shows the results of the DEC setting where the popularity of a data item is directly correlated with its size (popular items are large). The figure shows that SAAB-U improves utility by up to 30% compared to W-SJF. This increase in improvement, as compared with the previous experiment, is due to the fact that under this setting a policy which considers popularity (e.g., W-SJF) might give high priority to large-sized popular items, hence it will block the dissemination of small-size items which are associated with high utility values.

Figure 13 shows the results of the INC setting where the popularity of a data item is inversely correlated with its size (popular items are small). The figure shows that SAAB-U improves utility by up to only 9% (as opposed to 30% for the DEC setting). The reason is that under the INC settings, the popular items are also small. Hence, scheduling them will not have that prominent impact observed under the DEC setting.

Utility - Isolated Request Generation Setting: In this experiment we set the workload parameters as specified previously in our experiment on tardiness for the isolated request generation setting (Figure 10). However, in the experiment presented in this section, we measure utility as shown in Figure 14. The figure shows that SAAB-U achieves up to 13% improvement as compared with W-SJF, and up to 44% as compared with EDF at the high end of utilization. Also note that in this experiment we do not use any specific correlation between items popularity and their corresponding size.

VII. Related Work

Several research efforts have been directed towards broadcast scheduling under both the push-based and pull-based. For instance, the work in [1] introduces *Broadcast Disks*, a technique for improving the broadcast of non-uniformly accessed data in push-based broadcasting. Since power consumption is a major issue under the push-based model, power conservation indexing methods for single-attribute and multiple-attribute queries have been also proposed (e.g., [13], [12]).

On-demand data broadcasting has been shown to be more efficient than push-based broadcasting for unknown or frequently changing data access patterns [5]. This motivated proposing several on-demand broadcast scheduling policies for minimizing either response time (e.g., [5], [3], [4]), or minimizing the drop ratio for requests with hard deadlines (e.g., [22], [15], [8]).

For instance, towards minimizing respose time, the work in [4] propose RxW scheduling policy, which is a combination of MRF and First-Come-First-Served that achieves a good balance between the average and worst-case response time of the system. Hybrid solutions that combine push and on-demand data broadcasting have been also analyzed (e.g., [2], [21]). These approaches combine the scalability of the push model together with the ability of the on-demand model to adapt to changes in workload. A quantitative comparison between push and on-demand models is introduced in [6].

Towards minimizing request drop rate, [22] proposes SIN-α, an approach which considers request deadline and popularity in assigning the request priority. However, SIN-α assumes that all items are of the same size. The work in [15] extends SIN-α by considering variable data sizes, the first feasible deadline and the popularity of data items. Similarly, [8] considers item size in addition to item popularity and the current request slack. However, all of the above approaches assume hard deadlines where a request is dropped if its deadlines has passed.

Scheduling under soft deadlines is also studied in the context of database transactions (e.g., [20], [10]). However, in that conext each transaction is submitted by a single user, hence, there is no popularity involved in scheduling.

The closest work to ours is introduced in [9] which considers on-demand data broadcasting for requests with soft deadlines. In particular, [9] proposes combining heuristic based scheduling algorithms such as EDF or HUF (Highest Utility First) with local search techniques so that to improve the schedules generated by the heuristics. Additionally, [9] proposes a genetic algorithm, (2+1)-ES, which provides better performance than the local search methods as it can sample the search space more diversely. However, this typically results in a nondeterministic scheduler where the quality of the scheduling decision directly depends on the number of iterations available for running the algorithm. Compared to the work in [9], we show that efficient heuristic methods, such as our proposed SAAB, can achieve consistent and significant improvements over traditional heuristics at no additional overhead such as that incurred by genetic algorithm methods.

VIII. Conclusions

In this paper, we proposed SAAB, a new data broadcast scheduling policy for optimizing the performance of data broadcast systems in the presence of SLAs. SLAs present a challenge to the design of current broadcast systems since existing scheduling policies are tuned mainly to optimize either request response time or drop rate. However, under the SLA-based performance measures, we showed that the performance of such existing policies is highly sensitive to workload conditions. This motivated us to propose SAAB which is an SLA-aware adaptive broadcast scheduling policy. In particular, we proposed SAAB for optimizing the simple form of SLA known as tardiness. Further, we extended SAAB to consider the more general form of SLA defined in terms of utility functions. Our experimental evaluation shows that for both forms of SLA, SAAB consistently outperforms existing broadcast scheduling policies under all workload conditions.

Acknowledgments

The second author is supported in part by the Ontario Ministry of Research and Innovation Postdoctoral Fellowship. The authors would like to thank the anonymous reviewers for their insightful feedback. We would also like to thank Daniel Lupei for his comments and suggestions.

References

[1] Swarup Acharya, Rafael Alonso, Michael Franklin, and Stanley Zdonik. Broadcast disks: data management for asymmetric communication environments. In *SIGMOD*, 1995.

[2] Swarup Acharya, Michael Franklin, and Stanley Zdonik. Balancing push and pull for data broadcast. In *SIGMOD*, 1997.

[3] Swarup Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: new metrics and algorithms. In *MobiCom*, 1998.

[4] Demet Aksoy and Michael Franklin. R $\times$ W: a scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Trans. Netw.*, 7(6):846–860, 1999.

[5] Demet Aksoy and Michael J. Franklin. Scheduling for large-scale on-demand data broadcasting. In *INFOCOM*, 1998.

[6] Demet Aksoy and Mason Sin-Fai Leung. Pull vs push: a quantitative comparison for data broadcast. In *GLOBECOM*, 2004.

[7] Nikhil Bansal and Kedar Dhamdhere. Minimizing weighted flow time. *ACM Trans. Algorithms*, 3(4), 2007.

[8] Weiwei Cao and Demet Aksoy. MAI: multiple attributes integration for deadline-aware pull-based broadcast scheduling. In *ISCN*, 2006.

[9] Rinku Dewri, Indrakshi Ray, Indrajit Ray, and Darrell Whitley. Optimizing on-demand data broadcast scheduling in pervasive environments. In *EDBT*, 2008.

[10] Shenoda Guirguis, Mohamed A. Sharaf, Panos K. Chrysanthis, Alexandros Labrinidis, and Kirk Pruhs. Adaptive scheduling of web transactions. In *ICDE*, 2009.

[11] Sohail Hameed and Nitin H. Vaidya. Efficient algorithms for scheduling data broadcast. *Wirel. Netw.*, 5(3):183–193, 1999.

[12] Q. Hu, W.-C. Lee, and Dik Lun Lee. Power conservative multi-attribute queries on data broadcast. In *ICDE*, 2000.

[13] Tomasz Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air: Organization and access. *IEEE Trans. Knowl. Data Eng.*, 9(3):353 – 372, 1997.

[14] E. Jensen, C. Locke, and H. Tokuda. A time-driven scheduling model for real-time operating systems. In *RTSS*, 1985.

[15] Victor C. Lee, Xiao Wu, and Joseph Kee-Yin Ng. Scheduling real-time requests in on-demand data broadcast environments. *Real-Time Syst.*, 34(2):83–99, 2006.

[16] Simone Lupetti and Dmitrii Zagorodnov. Data popularity and shortest-job-first scheduling of network transfers. In *ICDT*, 2006.

[17] Kinji Matsumura, Kazuya Usui, Kenjiro Kai, and Koichi Ishikawa. Location-aware data broadcasting: an application for digital mobile broadcasting in Japan. In *MULTIMEDIA*, 2003.

[18] Stefan Parkvall, Eva Englund, Magnus Lundevall, and Johan Torsner. Evolving 3G mobile systems: broadband and broadcast services in WCDMA. *IEEE Communications Magazine*, 44(2):30–36, 2006.

[19] Mohamed A. Sharaf and Panos K. Chrysanthis. On-demand data broadcasting for mobile decision making. *Mob. Netw. Appl.*, 9(6):703–714, 2004.

[20] Mohamed A. Sharaf, Shenoda Guirguis, Alexandros Labrinidis, Kirk Pruhs, and Panos K. Chrysanthis. Poster session: Asets: A self-managing transaction scheduler. In *ICDE Workshops*, 2008.

[21] Konstantinos Stathatos, Nick Roussopoulos, and John S. Baras. Adaptive data broadcast in hybrid networks. In *VLDB*, 1997.

[22] Jianliang Xu, Xueyan Tang, and Wang-Chien Lee. Time-critical on-demand data broadcast: Algorithms, analysis, and performance evaluation. *IEEE Trans. Parallel Distrib. Syst.*, 17(1):3–14, 2006.