Scheduling Real-Time Data Items In Multiple Channels And Multiple Receivers Environments

Guanling Lee, Yi-Ning Pan and Arbee L.P. Chen
Department of Computer Science and Information Engineering
National Dong Hwa University
Hualien, Taiwan 973, R.O.C.
Email: guanling@mail.ndhu.edu.tw

Abstract

In the real-time environments, information is disseminated to clients with timing constraint. In this paper, we focus on the real time data scheduling problem in multiple broadcast channels environments where the clients equip with multiple receivers. Each request is associated with a deadline. The clients can retrieve data items either from the broadcast channels or make requests to the server and then listen to the broadcast channels. The intention of our work is to serve as many requests as possible.

In our approach, the content of the broadcast program is first decided. Afterwards, the periodic broadcast program generation algorithm and the on-demand broadcast program generation algorithm are proposed. Simulations are performed to justify the benefit of our approaches. We conclude that our data scheduling algorithms are scalable and the time spent on executing our data scheduling algorithms is low. Moreover, by using our data scheduling algorithms, the percentage of requests that miss their deadlines is also low.

KEYWORDS: real-time database, timing constraint, periodic broadcast program, on_demand broadcast program, multiple broadcast channels.

1. INTRODUCTION

Rapid advances in wireless communications and software/hardware technologies enable a client carrying a mobile device to access information without the restriction of time and location. Broadcast-based information systems provide the dissemination of information with a cost independent of the number of clients, which compensates for limited bandwidth in the wireless environments.

In the broadcast data delivery environments, there are two basic modes for data transmission. In the first mode, the *pull mode*, the clients request data items from the server via the *uplink channel*. The server determines which request to be satisfied next and then disseminates the required data item via the *broadcast channel.* In [5], many traditional

pull-based scheduling strategies are compared. In the *First Come First Served* scheduling strategy, requests are served in the order of their arrival time. However, popular data items are requested more frequently, the same data item may disseminate to the clients repetitively which leads to waste of bandwidth. In the *Most Request First* scheduling strategy, the data item with maximum number of requests is selected for broadcast. Therefore, requests on unpopular data items may have very long waiting time or may never be served. To avoid the situation of starvation, the *Longest Wait First* scheduling strategy is proposed. Besides, in [5], they propose a promising scheduling strategy, called RxW, which considers not only the number of requests for each object but also the amount of waiting time of the oldest requests for the data item. Moreover, the effectiveness of the batch scheme on some pull-based scheduling strategies has also been examined in [4].

In the second mode, the *push mode*, the server transmits pre-selected data item set via the broadcast channels according to the *broadcast program.* The broadcast program determines the order of data items in the broadcast channels. The clients can retrieve broadcast data by tuning to the broadcast channels. The simplest way to organize the broadcast data items is the *flat* organization [6]. In the flat organization, the server broadcasts the union of data items needed by clients periodically. However, access frequency of each data item is not the same in reality. Therefore, in [8] [9] [15], the *non-flat* organization method that broadcasts the frequently accessed data items more times than those unpopular ones are proposed. Some researches have focus on generating broadcast program on multiple channels. In [18], the data items together with the index of them forms an index tree. The index nodes on each level of the index tree are assigned to an individual channel. The client tunes into the first channel to get the root of the index tree and then follows the index pointers to retrieve the desired data item. Therefore, the number of channels needed to allocate the index nodes is decided by the depth of the index tree, which is not flexible. Allocating each index node in an individual channel presents waste of space. An algorithm that avoids these drawbacks is proposed in [14]. This algorithm tries to find the optimal index and data allocation to minimize the average access latency for any number of broadcast channels. In [19], a data scheduling algorithm which considers index and data items replication in the multiple broadcast channels environment is proposed. In [15], a log-time algorithm for distributing the broadcast data items over multiple broadcast channels is proposed, and this algorithm is modified for transmitting the data items with errors. A near optimal algorithm for generating broadcast program over multiple broadcast channels without indexing structure is proposed in [20].

There is an integrated mode, called the *hybrid mode*, which combines the push and pull modes together. In [7], data items are classified into the broadcast or the on-demand categorizes. The server multiplexes the push and pull data for dissemination. Therefore, clients can either listen to the broadcast channels to retrieve the desired data items or submit requests via the uplink channel and then wait for the data items to be transmitted in the on-demand broadcast channel.

Previous researches on broadcast scheduling assume that once a client generates a request, this request will not be discarded until it is satisfied. However, in real application, the request may associate with a deadline. For example, if a client needs the traffic information to decide which road to drive, he must receive the response before he reaches the crossroad. Some recent works begin to address the transmission of the real-time database. The *pinwheel scheduling problem* which tries to schedule the real-time database for satellite-based communication was first introduced in [10]. This problem is generalized in [11] which can be applied to generate the fault-tolerant, real-time broadcast disks [3]. Since impatient users may withdraw their requests before they are served, a data broadcast scheduling algorithm to those users is proposed in [1]. Moreover, an adaptive hybrid transmission technique based on the PinOpt [11] algorithm is presented in [2] and [17]. In this model, information is disseminated to clients with timing constraint. This algorithm allocates data items in single broadcast channel, which motivates us to extend it to the multiple broadcast channels environments.

In this paper, the problem of transmitting data items with timing constraint in the multiple broadcast channels where the clients equip with multiple receivers is studied. In our approach, the data items are divided into two sets, the *broadcast data set* and the *on-demand data set.* The data items in the broadcast data set is periodically broadcast on the broadcast channels while the data items in the on-demand data set are transmitted when they are requested. The server offline allocates the data items in the broadcast data set on the multiple broadcast channels. The clients retrieved data items from the broadcast channels are guaranteed to receive them within the timing constraint. Once the desired data item is not included in the broadcast channels, clients make requests associated with deadlines to the server. By using the bandwidth remaining for the on-demand mode, the server broadcasts the requested data items in an online fashion.

The rest of this paper is organized as follows. In Section 2, the scheduling problem is formulated and the system architecture is introduced. The scheduling algorithms for the broadcast data set and the on-demand data set are presented in Section 3. In Section 4, a

simulation model and the analysis of the simulation results are described. Finally, Section 5 concludes this work.

2. WIRELESS DATA DELIVERY MODEL

In this section, the real-time scheduling problem and the system architecture are introduced.

2.1 Problem Formulation

Database in the server is constituted by many data items. The data items may have various sizes and is conceptually split into several *pages*. Assume the time needed to broadcast a single data page is called one *time slot*. Each data item X is characterized by X(X.s , X.p). X.s is the *size* of the data item X which is equal to the number of pages of data item X. X.p is the *period* of data item X which is the timing constraint of data item X. That is, the number of time slots needed to be accessed for receiving data item X can not exceed X.p. Therefore, data item X in the broadcast data set will be broadcast X.s pages every X.p consecutive time slots to promise that a client needs data item X in the broadcast data set is guaranteed to receive at least X.s pages of data item X in the period of time not greater than X.p. For the on-demand mode, a client makes a request associated with a deadline to the server, and then listens to the broadcast channels until the desired data item is broadcast or until the acceptable waiting time is exceeded. The relationship between the size and the period of data item X is $X.s \leqq X.p*m$, where m is the number of the broadcast channels.

The performance metric of our problem is not to minimize the average access time but the percentage of information demands that is satisfied by the server.

2.2 System Architecture

Server side: The broadcast server is divided into two modes, the broadcast mode and the on-demand mode. Each data item in the database is classified in one of these two modes, denoted as broadcast data set and on-demand data set, respectively. The fraction of bandwidth allocates to these two modes is dynamically decided by the density of the broadcast data set. Moreover, the bandwidth for broadcast is divided into m+1 channels. One of the channels is reserved for delivery *broadcast schema*. The broadcast schema contains the information of the broadcast data set. The broadcast scheduler generates periodical broadcast program on the m broadcast channels. Because the clients equip with multiple receivers, pages of a data item can appear in more than one channel at the same time. By using the bandwidth remaining for the on-demand mode, the on-demand scheduler selects the request

to be served and broadcasts the requested data items in an online fashion.

Client side: Each client can require one data item per request associated with a deadline constraint. When a client needs a data item, it first tunes in the broadcast channels to retrieve the broadcast schema. By examining the broadcast schema, the client can determine whether he can get the data item from the broadcast channels. If the needed data item is in the broadcast data set, the client tunes in the broadcast channels and retrieves the desired data item. Otherwise, the client sends a request to the server via the uplink channel, and listens to the broadcast channels to retrieve the data pages. Assume there are m broadcast channels available in the server side and each client equips with m receivers. Therefore, a client will receive m data pages simultaneously, and then prunes the irreverent data pages until all the pages of desired data item is retrieved or until the acceptable waiting time exceeded. Since the time of each client tunes in the broadcast channels is unpredictable, a client may receive page 5 first, and then page 1, page 2, page 3, and page 4. Therefore, the clients must have the ability of reordering data pages themselves.

3. DATA SCHEDULING

In this section, the issue of data scheduling in both modes are discussed. Based on Pinfair algorithm presented in [13], we propose a periodical broadcast program generation algorithm. Moreover, five request insertion policies are also proposed when developing the on-demand scheduling algorithm.

3.1 Preliminary

In this section, we introduce the generalized pinwheel task scheduling problem in [13] which we employee to generate our broadcast program. The model of the generalized pinwheel task system is as follows: *Given a multiset* $\{(a_1, b_1),(a_2, b_2),\ldots,(a_n, b_n)\}$ *of ordered pairs of positive integers, determine an infinite sequence over the symbols* $\{1, 2, 3,\ldots, n\}$ *such that, for each i,* $1 \leq i \leq n$, *any subsequence of* b_i *consecutive symbols contains at least* a_i *i's*. Each generalized pinwheel task X is characterized by a computation requirement X.a and a window size X.b. The weight of a task X is $\frac{X.a+1}{X.b}$ and is denoted by X.w. Given a feasible task set, the Pinfair algorithm can constructs a pinwheel schedule in single resource environments, that is, for each task X in the feasible task set, it is expected to allocate the shared resources for at least X.a times out of every X.b consecutive time slots.

We observe that the generalized pinwheel scheduling problem is similar to the SCSR (*single channel with single receiver*) broadcast scheduling problem. Each data item X can be viewed as a task X, where X.s and X.p are mapped to X.a and X.b respectively. Moreover, the single broadcast channel is mapped to the single shared resource. By transforming the generalized pinwheel scheduling problem to the broadcast scheduling problem, the Pinfair algorithm can be applied to solve the SCSR broadcast scheduling problem, that is, if data item X is broadcast in the broadcast channel, it is expected to be broadcast for at least X.s pages out of every X.p consecutive time slots.

The definitions of the terms which is used to introduce the Pinfair algorithm are as follows:

- *allocated (X, t)* = the number of resources allocated to a task X during the interval (0, t).
- *earliest (X, j)* = the earliest time a task X can be scheduled the jth times, and is defined by $\left\lfloor \frac{j}{X.w} \right\rfloor$.
- *latest (X, j)* = the latest time a task X must be scheduled the jth times, and is defined by $\left\lceil \frac{j+1}{X.w} \right\rceil - 1$.
- A task X is said to be *contending* if allocated *(X, t)*=k and earliest *(X, k)* $\leqq$ t. The pseudo-deadline X.d is defined as *latest (X, k)*.

There are two problems when developing the scheduling algorithm [12]. One is the *decision problem* and the other is the *scheduling problem*. The decision problem is to determine whether or not a given instance is feasible. The scheduling problem is to actually construct the schedule for a given feasible instance. The following theorem proposed in [13] solves the decision problem while the Pinfair algorithm is the solution of the other problem.

Theorem 1. Any system of generalized pinwheel tasks Γ satisfying $\sum_{X \in \Gamma} X.w \leq 1$, where X.w is the weight of a task X and is defined as $\frac{X.s+1}{X.p}$, can be successfully scheduled by Algorithm Pinfair.

The *Pinfair algorithm* is described below:

Algorithm Pinfair (Γ), where Γ is a system of pinwheel tasks.
Step1: For each task X ∈ Γ, define a weight X.w as $\frac{X.s+1}{X.p}$.
Step2: If $\sum_{X \in \Gamma} X.w > 1$, return failure.
Step3: Allocate the processor at each time slot to the contending task with the tightest pseudo-deadline.

In our approach, the problem of transmitting data items with timing constraint in the multiple channels environments where the clients equip with multiple receivers is considered. We formulate our problem as the *MCMR* (*Multiple Channels with Multiple Receivers*) broadcast scheduling problem, and will be discussed in the following.

3.2 Periodical Broadcast Program Generation

To solve MCMR broadcast schedule problem, we first transform it to SCSR broadcast schedule problem. By solving SCSR broadcast schedule problem, the MCMR broadcast schedule problem will be solved. In subsection 3.2.1, how to transform the MCMR broadcast schedule problem to the SCSR one is presented. How to determine the broadcast data set is discussed in subsection 3.2.2. The periodical broadcast program generation algorithm is presented in subsection 3.2.3.

3.2.1 Problem Transformation

In our approach, the MCMR broadcast scheduling problem is first mapped to the SCSR one. Before scheduling, the time slots C_{nm} in the multiple broadcast channels are mapped to a consecutive time slots in the single broadcast channel. The mapping sequence is C_{01} C_{02} … C_{0m} C_{11} C_{12} …C_{nm}, where n is time T, and m is the number of broadcast channels. **Figure 1** shows the graph representation of the mapping sequence. Moreover, each data item X(X.s , X.p) in the MCMR environments is mapped to X(X.s , X.p*m) in the SCSR environments. After constructing the broadcast program, the time slots are mapped to their original position.

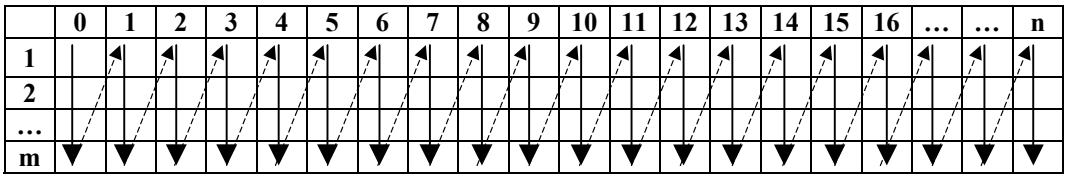


Figure 1: The Graph Representation of the Mapping Sequence

3.2.2 Determine the Broadcast Data Set

When transforming the MCMR broadcast scheduling problem to the SCSR broadcast scheduling problem, the data item X(X.s, X.p) in the MCMR environment is mapped to X(X.s, X. p*m) in the SCSR environment. To satisfy Theorem 1, the *weight* of data item X is defined as $\frac{X.s+1}{X.p*m}$, where m is the number of channels.

Lemma 1. The weight of data item X mapped from the MCMR scheduling problem to the SCSR scheduling problem is defined as $X.w = \frac{X.s+1}{X.p*m}$.

Proof: The worst case of the scheduling algorithm is that the jth times data item X being scheduled in the earliest time slot, and for the next allocation, said (j+1)th allocation of data item X, the slot is the latest time slot. Since a client is guaranteed to receive at least X.s pages

for every X.p*m consecutive slots, the interval of latest (X , j+X.s) – earliest (X , j) can never exceed X.p*m, that is

$$
\begin{aligned}
&latest(X, j+X.s) - earliest(X, j) \\
&= \left\lceil \frac{j+X.s+1}{X.w} \right\rceil - 1 - \left\lfloor \frac{j}{X.w} \right\rfloor \\
&= \left\lceil \frac{j}{X.w} + \frac{X.s+1}{X.w} \right\rceil - \left\lfloor \frac{j}{X.w} \right\rfloor - 1 \\
&= \left\lceil \frac{j}{X.w} \right\rceil + \frac{X.s+1}{\left(\frac{X.s+1}{X.p*m} \right)} - \left\lfloor \frac{j}{X.w} \right\rfloor - 1 \\
&\le X.p*m + 1 - 1 \\
&= X.p*m
\end{aligned}
$$

■

After defining the weight of each data item, we address the problem of selecting the optimal broadcast data set. The broadcast data set is optimal if these two condition hold:

1. $\sum_{X \in Broadcast\,Data\,Set} X.w \le 1$. (According to Theorem 1)
2. $\sum_{X \in Broadcast\,Data\,Set} access\ frequency\ of\ data\ item\ X$ is maximum.

Solving this problem is the same as solving the 0-1 knapsack problem. Although 0-1 knapsack problem is a NP-complete problem, there are many approximation algorithm to solve it. The detail of the algorithm is omitted here. After computing the optimal broadcast data set, the remaining data items in the database server not included in the broadcast data set

Algorithm Broadcast Data Set Selection (Γ)
Step1: (The Pruning Step)
For any data item X in the previous broadcast data set, if the access frequency of data item X is less than $\frac{previous\ broadcast\ cycle\ length}{X.p}$, then do not classify data item X in the broadcast data set.
Step2: Solve the 0-1 knapsack problem.
Step3: Classify the remaining data items in the on-demand data set.

are said to be in the on-demand data set. Following is the description of the *Broadcast Data Set Selection Algorithm*:

For each data item X, we check whether it is worth broadcasting in the pruning step. The purpose of the pruning step is to prune those data items whose requests are fewer than the number of times they are broadcast in the previous broadcast cycle. For example, if we broadcast a data item 10 times in the previous broadcast cycle, and it is requested only 5 times, it will not be included in the next broadcast program.

Moreover, the *broadcast cycle length* is determined by the least common multiple of all the periods of the data items in the broadcast data set. The percentage of bandwidth allocated

to the broadcast mode is $\sum_{X \in Broadcast\ Data\ Set} X.w$, and the percentage of bandwidth allocated to the on-demand mode is 1- $\sum_{X \in Broadcast\ Data\ Set} X.w$.

3.2.3 The Periodical Broadcast Program Generation Algorithm

The procedures of the *Broadcast Program Generation Algorithm* are as follows:

Algorithm Broadcast Program Generation
Step1: Map each time slot in the MCMR environments to the SCSR environments.
Step2: Select an optimal broadcast data set.
Step3: For each time T, allocate the time slot to the contending data item in the broadcast data set with the tightest pseudo-deadline.
Step4: Map each time slot in the SCSR environments to the MCMR environments.
Step5: For each time T, record the number of unallocated time slots in num(T).

For instance, assume there are three channels available. Consider an optimal broadcast data set contains A, B, C, and D and each of the item are characterized by {A(21 , 20), B(4 , 20), C(9 , 10), D(1 , 40)}. Thus, the broadcast cycle length is 40. After transformation, the broadcast data set is {A(21 , 60), B(4 , 60), C(9 , 30), D(1 , 120)}, and the broadcast cycle length is 120. Followings are the processes of generating the broadcast program in the SCSR environments.

Slot	**0**	**1**	**2**	**3**	**4**
Data Item	A_1	C_1	A_2	C_2	B_1

k = allocated (A,t)	1	1	2	2	2
(earliest(A,k),latest(A,k))	(0,**2**)		(2,**5**)		
k = allocated (B,t)	0	0	0	0	1
(earliest(B,k),latest(B,k))	(0,11)	(0,11)	(0,11)	(0,11)	(0,**11**)
k = allocated (C,t)	0	1	1	2	2
(earliest(C,k),latest(C,k))	(0,2)	(0,**2**)		(3,**5**)	
k = allocated (D,t)	0	0	0	0	0
(earliest(D,k),latest(D,k))	(0,59)	(0,59)	(0,59)	(0,59)	(0,59)

In T=0, all the data items are in the contending mode, thus, we allocate the 0th time slot to the contending data item A with the smallest pseudo-deadline 2. In T=1, we observe that only data item B, C and D are in the contending mode. Comparing the pseudo-deadlines of these three data items, we allocate the 1th time slot to data item C. As we proceed with this procedure, a broadcast program with length 120 is generated. Mapping the resultant broadcast program from the SCSR environments to the MCMR environments, we get the following broadcast program with length 40:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
1	A_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	A_{11}	A_{12}	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_1	A_1

2	C_1	B_1	D_1	A_5	A_6	A_7	A_8	A_9	A_{10}	C_1	C_2	A_{14}	B_4	A_{16}	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}	C_2
3	A_2	A_3	A_4		B_2				B_3		A_{13}		A_{15}				B_1			D_1

	20	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	33	**34**	**35**	**36**	**37**	**38**	**39**
1	A_2	C_4	C_5	C_6	C_7	C_8	C_9	C_1	C_2	A_{12}	A_{13}	C_5	C_6	C_7	C_8	C_9	C_1	C_2	C_3	A_2
2	C_3	B_2	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	C_3	C_4	A_{15}	A_{16}	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}	A_1	C_4
3	A_3	A_4			B_3				B_4		A_{14}		B_1				B_2			

The broadcast program perfectly meets our assumptions. The clients need data item C will receive C_1, C_2, …, C_9 in at most ten time slots regardless of what time clients start listening to the broadcast channels. Moreover, in each time t, the number of empty slots is stored in *num(t)*. Therefore, $0 \le \text{num(t)} \le \text{m}$. The server records the num list of the broadcast program to facilitate the on-demand scheduling algorithm.

When the broadcast program cannot satisfy the client's request, an explicit request can be made to the server about the data item needed associated with the tolerable deadline via the uplink channel. The unallocated time slots are used to transmit the required data items.

3.3 On-Demand Program Generation

The main issue of developing an on-demand scheduling algorithm is the scalability. That is, the scheduling overhead of processing new arrival request and selecting the next served request must be reduced [16]. When new requests arrive, the server inserts them into a *priority queue PQ* according to the priorities determined by the *request insertion policies*. The on-demand scheduler selects the request with the highest priority and then broadcast the data item of interest. Assume each request is of the form of X(X.page_set , X.deadline), where X.page_set is the demand page set of data item X and X.deadline is the timing constraint of the request. The number of pages in X.page_set is denoted by X.psize. Moreover, the constraint that X.psize$\leqq$X.deadline*m must be met. The priority of each request is determined by the following *On-Demand Request Insertion Policies*:

Policy 1: *Earliest Deadline First (EDF)*

$$priority \ \alpha \ \frac{1}{X.deadline}.$$

Policy 2: *Smallest Size First (SSF)*

$$priority \ \alpha \ \frac{1}{X.psize}.$$

Policy 3: *Largest Size First (LSF)*

$$priority \ \alpha \ X.psize.$$

Policy 4: *Smallest Difference First (SDF)*

$$priority\ \alpha\ \frac{1}{X.deadline * m - X.psize}.$$

Policy 5: *Max Frequency Times Difference First (MFTDF)*

$$priority\ \alpha\ \frac{Access\ Frequency\ \ of\ \ data\ \ item\ X}{X.deadline * m - X.psize}.$$

There are three factors considered when developing the priority of each request, the deadline, the size of the demand page set, and the access frequency. The priority in the first three request insertion policies is determined by only one factor. However, in the SDF request insertion policy, the priority is decided by the difference of X.deadline and X.psize, which can be viewed as the *tightness* of the request. Assume that the current time is t, m is the number of receivers. If we do not broadcast the first demand page of data item X during time $\left[t, \left\lfloor \frac{t*m+b*m-X.deadline}{m} \right\rfloor\right]$, we should drop this request from PQ. Therefore, the smaller the value is, the higher the priority of this request should be set. On the other hand, many requests may require the same data item in a very short duration. Transmit one copy of this data item will satisfy all requests. The MFTDF request insertion policy considers the access frequency of each data item together with the deadline and the size of the demand page set. The requests are inserted into PQ according to their priorities defined by each request insertion policy.

After constructing the broadcast program, the empty time slots in each time unit are used by the on-demand mode. It is obvious that not every request can be served. Therefore, a *Request Selection Algorithm* that selects a request from PQ as the *processing request* is proposed. The detail of the algorithm is as the following:

The *Request Selection Algorithm*:

Algorithm Request Selection
For each nonzero num(t) and PQ not empty ,
Step1: Select the first request X(X.page_set , X.deadline) in PQ.
Compute num_sum = num(t) + num(t+1) + … + num(t-X.deadline-1).
B1: If num_sum＜X.psize, then remove the request from PQ. Go to Step 1.
B2: If num_sum≧X.psize, then find the minimum t' such that X.psize≦num(t) + num(t+1) + …+ num(t').
Allocated slots=num(t)+…+num(t'-1).
Set num(t)～num(t'-1) to zero. /*allocate num(t)~num(t'-1) to X */
num(t')=num(t')-(X.psize-allocated slots)./*calculate the remaining available slots in num(t')*/
Remove the request from PQ and set this request as the processing request.

The idea of this algorithm is that for every nonzero num(t), we select the first request X and check if the total number of empty slots in the period [t, t+X.deadline-1] is greater than or

equal to the size of the demand page set. If the condition holds, the server broadcasts the demand pages. Otherwise, remove this request from PQ and examine the next request.

For example, a part of the num list of the generated broadcast program in the previous example is:

Time	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
num	0	0	0	1	0	1	1	1	0	1	0	1	0	1	1	1	0	1	1	0	0

Assume the EDF request insertion policy is used. Because there are no time slots available in T = 0~2, no request will be selected. In T = 3, the requests in PQ are $X(X_1 \rightarrow X_3$, 5), $Y(Y_1 \rightarrow Y_6$, 8) and $W(W_1 \rightarrow W_3$, 12). We check that num_sum(X) = num(3) + num(4) + … + num(7) = $4 > 3$, thus, $X(X_1 \rightarrow X_3$, 5) is set to be the processing request. After broadcasting X_1 in T=3, X_2 in T=5 and X_3 in T=6, the num list is modified as follows:

Tlime	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
num	0	0	0	0	0	0	0	1	0	1	0	1	0	1	1	1	0	1	1	0	0

Afterwards, in T=7, there is one empty slot available. We select another request from PQ as the processing request, and proceeds with the procedures.

The clients may require the same data item as the processing request. Nevertheless, if some pages of the data item have been broadcast, the clients cannot retrieve all pages of the data item. One solution to this situation is to treat every request as a new one. Obviously, that will be a frightful waste of the bandwidth. This problem could be solved according to the following procedures. Compute the different set Δ of the demand page set of the requests in PQ which require the same data item as the processing request. If Δ is empty, and the deadline of the request is larger than that of the processing request, the requests with the same required data item in PQ can be dropped. This is because these requests can be satisfied by the processing request. If Δ is not empty, the demand page set of the requests with the same required data item in PQ are set to Δ and will be kept in PQ even if the deadlines are smaller than that of the processing request. We cannot exclude the possibility that these requests may be satisfied. Following is our on-demand scheduling algorithm.

The *On-Demand Scheduling Algorithm*:

Algorithm On-Demand Scheduling

Setp1: For each time t, remove requests with X.psize＞X.deadline*m in PQ.

Step2: B1: If num(t)=0, set t=t+1. Subtract 1 from the deadline of the processing request and each request in PQ. Go to step1.

B2: If num(t)>0, generate the processing request by the Request Selection Algorithm.

Select all requests in PQ with the same required data item as the processing request. Compute each difference set Δ of the demand page set between requests in PQ and the processing request. There are three conditions:

①Δ is ϕ, and the deadline of the request in PQ is greater than that of the processing request. Drop the request from PQ. /* can be satisfied by broadcasting processing request */

②Δ is not ϕ, and the deadline of the request in PQ is greater than or equal to that of the processing request. Keep the request in PQ, and set Δ as the demand page set of the request. /*part of the demand page set "can" be satisfied by broadcasting processing request */

③Δ is not ϕ, and the deadline of the request in PQ is smaller than the processing request. Keep the request in PQ, and set Δ as the demand page set of the request. /*part of the demand page set "may" be satisfied by broadcasting processing request */

We continue the previous example to illustrate the on-demand scheduling algorithm. **Figure 2** shows the graph representation of PQ and the processing request. T(before) means the state of PQ in time T before allocation. The state of PQ after allocation is denoted as T(after). $X(X_b \rightarrow X_e, d)$①, X_b and X_e indicate the beginning and end page of the demand pages of item X. d is the deadline of the request. ① means that the data item satisfies the 1th condition in B2 of Step 2 in the on-demand scheduling algorithm. **$Y(Y_b \rightarrow Y_e, f)$** is the new request arrives in time T and is denoted by boldface. ~~$W(W_b \rightarrow W_e, g)$~~ means that this request is dropped for a certain reason. In this example, the request insertion policy is the EDF.

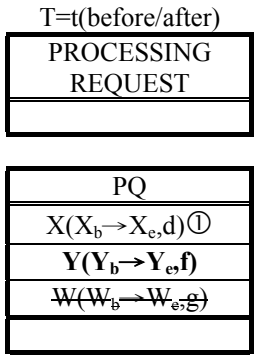


Figure 2: The Graph Representation of the PQ and the Processing Request.

In the following, a part of the resultant broadcast program and the on-demand program is shown. (The time slots with shadows are used as on-demand transmission)

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
1	A_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	A_{11}	A_{12}	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_1	A_1
2	C_1	B_1	D_1	A_5	A_6	A_7	A_8	A_9	A_{10}	C_1	C_2	A_{14}	B_4	A_{16}	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}	C_2
3	A_2	A_3	A_4	X_1	B_2	X_2	X_3	X_1	B_3	X_2	A_{13}	…	A_{15}	…	…	…	B_1	…	…	D_1

The step by step results with explanations of the on-demand scheduling algorithm is depicted as follows:

T=0	T=1	T=2	T=3(before)	T=3(after)
PROCESSING REQUEST	PROCESSING REQUEST	PROCESSING REQUEST	PROCESSING REQUEST	PROCESSING REQUEST
			$X(X_1 \rightarrow X_3,9)$	$X(X_2 \rightarrow X_3,9)$
PQ	PQ	PQ	PQ	PQ
$X(X_1 \rightarrow X_3,12)$	$X(X_1 \rightarrow X_3,11)$	$X(X_1 \rightarrow X_3,10)$	~~$X(X_1 \rightarrow X_3,10)$~~ ①	$Y(Y_1 \rightarrow Y_3,12)$
$Y(Y_1 \rightarrow Y_3,15)$	$Y(Y_1 \rightarrow Y_3,14)$	$Y(Y_1 \rightarrow Y_3,13)$	$Y(Y_1 \rightarrow Y_3,12)$	$Z(Z_1 \rightarrow Z_2,23)$
	$Z(Z_1 \rightarrow Z_2,25)$	$Z(Z_1 \rightarrow Z_2,24)$	$Z(Z_1 \rightarrow Z_2,23)$	

In T=0~2, there is no empty slots, therefore, no request in PQ will be served. In T = 3 (before allocation), $X(X_1 \rightarrow X_3, 9)$ is selected to be the processing request and the new arrival request $X(X_1 \rightarrow X_3, 10)$ will be deleted since it can be satisfied by the processing request.

T=4	T=5(before)	T=5(after)	T=6(before)	T=6(after)
PROCESSING REQUEST	PROCESSING REQUEST	PROCESSING REQUEST	PROCESSING REQUEST	PROCESSING REQUEST
$X(X_2 \rightarrow X_3,8)$	$X(X_2 \rightarrow X_3,7)$	$X(X_3,7)$	$X(X_3,6)$	
PQ	PQ	PQ	PQ	PQ
$Y(Y_1 \rightarrow Y_3,11)$	~~$X(X_1 \rightarrow X_3,8)$~~ ②	$X(X_1,8)$	~~$X(X_1 \rightarrow X_3,4)$~~ ③	$X(X_1 \rightarrow X_2,4)$
$Z(Z_1 \rightarrow Z_2,22)$	**$X(X_1,8)$**	$Y(Y_1 \rightarrow Y_3,10)$	**$X(X_1 \rightarrow X_2,4)$**	$X(X_1,7)$
	$Y(Y_1 \rightarrow Y_3,10)$	$Z(Z_1 \rightarrow Z_2,21)$	$X(X_1,7)$	$Y(Y_1 \rightarrow Y_3,9)$
	$Z(Z_1 \rightarrow Z_2,21)$		$Y(Y_1 \rightarrow Y_3,9)$	$Z(Z_1 \rightarrow Z_2,20)$
			$Z(Z_1 \rightarrow Z_2,20)$	

In T = 5, consider the new arrival request $X(X_1 \rightarrow X_3, 8)$ and the processing request $X(X_2 \rightarrow X_3, 7)$. The difference set Δ is not empty, and the deadline of the former request is larger than that of the later one, therefore, $X(X_1 \rightarrow X_3, 8)$ can be replaced by $X(X_1, 8)$. In T=6, the difference set D of new arrival request $X(X_1 \text{->} X_3, 4)$ and the processing request $X(X_3, 6)$ is not empty. We replace $X(X_1 \rightarrow X_3, 4)$ with $X(X_1 \rightarrow X_2, 4)$. Although the deadline of the later request is smaller than that of the processing request, this request can be satisfied if it is selected as the processing request in T=7.

As proceeding with these procedures, the on-demand program is generated. Moreover, the steps of on-demand scheduling algorithm with other request insertion policies are similar to the EDF one, and are omitted here.

4. SIMULATION

4.1 Parameters Setting

The following table shows the simulation parameters:

Parameter	Default Value	Range
Uplink channel capacity	8 request / time unit	
Request arrival rate	Poisson distribution among 0~8 requests	0~8 request / time unit
Size of each item	Uniform distribution among 1~8 pages	1~8 pages
Number of channels	8 channels	3~10 channels
Number of data items	500	200~1000
RF value	0	0~1

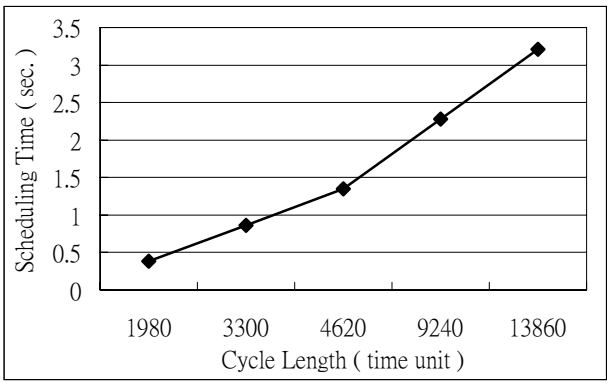
Table 1: Simulation Parameters

We consider a database consisting of 500 distinct data items with various sizes and periods. The period and size of each data item follow the uniform distribution and are permanently associated with each data item. The access frequency distribution of data items is modeled as Zipf distribution. Moreover, a workload generator generates on-demand requests associated with deadline constraints every time unit. The request arrival rate follows the Poisson distribution. The maximum capacity of the uplink channel is 8 requests per time unit.

Unlike the traditional broadcast scheduling algorithms that try to minimize the average access time, the metric to evaluate the performance of our scheduling algorithms is the percentage of deadlines of the requests that are missed.

4.2 Comparison of the Scheduling Time with Different Broadcast Cycle Length

After selecting a feasible, optimal broadcast data set, the broadcast program is generated offline. The broadcast cycle length is decided by the least common multiple of all the periods of data items in the broadcast data set. **Figure 3** shows the relationship between the time needed to generate the broadcast program verses different broadcast cycle length.



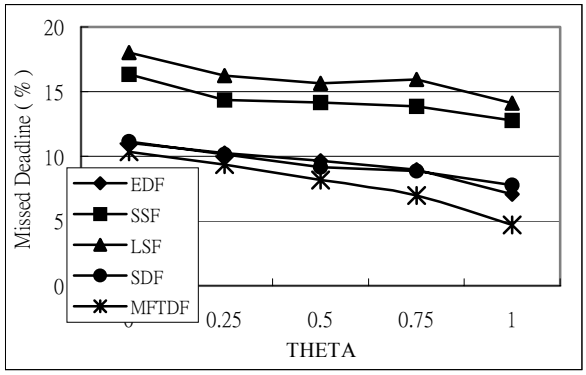


Figure 3: The Scheduling Time versus Different Broadcast Cycle Length.

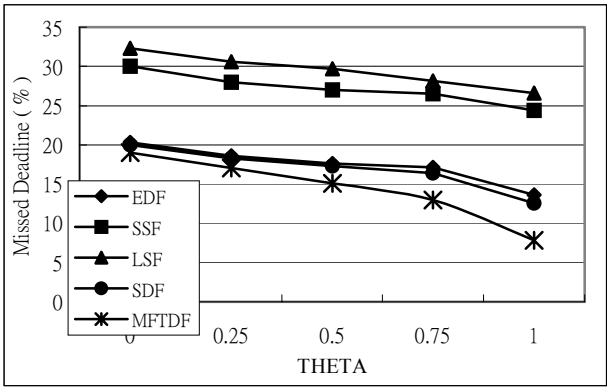
Figure 4: The Missed Deadline Over All Requests versus Different Access Skew Coefficient.

It can be seen that as the length increases, the time needed to compute the broadcast program increases.

4.3 Comparison of Different Request Insertion Policies

To evaluate the performance of five request insertion policies, two simulation-based experiments are carried out.

In the first simulation, we examine the effect of different access frequency distribution. The access frequency of each data item follows the zipf distribution which can be expressed as $P_i = \frac{\left(1/i\right)^{\theta}}{\sum_{i=1}^{M}\left(1/i\right)^{\theta}}$, $1 \le i \le M$, where M is the number of data items and θ is the *access skew coefficient*. For $\theta = 0$, the Zipf distribution reduces to the uniform distribution. As θ increases, the distribution becomes more skewed which means the range of p_i becomes larger. By tuning the access skew coefficient, we get different access frequency models. **Figure 4** and **Figure 5** plot the percentage of deadline missed over all requests and over on-demand requests versus different access skew coefficient. From the simulation results, we observer that as θ increases, more requests will be served. This is because when requests are targeted to fewer data items, transmission of those hot data items will satisfy more requests. However, for $\theta = 0$, there will be more requests sent to the sever than the skewed model since data items are accessed with equal probability.



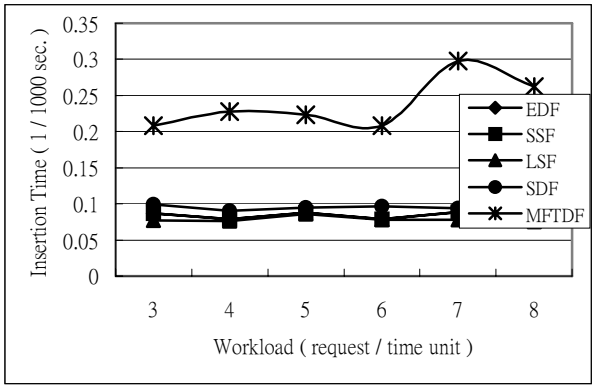


Figure 5: The Missed Deadline Over On-Demand Requests versus Different Access Skew Coefficient.

Figure 6: The Request Insertion Time versus Different Workload.

In the second simulation, we examine the effect of different workload. The workload of the server is determined by the number of requests arrived per time unit. The time needed to insert a new request into PQ is shown in **Figure 6**. The time spent in the EDF insertion policy

is shorter than all the other request insertion policies. The reason is that as the size of demand page set and the access frequency of requests changes, the order of all the requests in PQ will be affected in the other four request insertion policies. Since the insertion times are relative small, we can ignore it.

The percentage of deadline missed over all requests versus different workload is shown in **Figure 7** Moreover, the percentage of deadline missed over on-demand requests versus different workload is shown in **Figure 8**. We observe that the number of requests that cannot be satisfied increases as the workload of the server increases.

In the request insertion policies, three factors, the deadline, the size of the demand page set, and the access frequency of each request, are considered when determining the priority of each request. In the SSF request insertion policy, requests with smaller demand page set will be served earlier. However, if many requests require large demand page set with tight deadline, there will be more chance to discard them. This situation is also true for the LSF request insertion policy. In the SDF request insertion policy, the tightness of each request is considered and the performance is as good as the EDF one. Moreover, in the MFTDF request insertion policy, all three factors are combined together which leads to better performance in the simulation results.

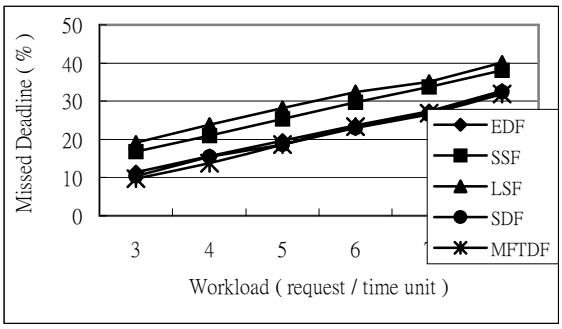


Figure 7: The Missed Deadline Over All Requests versus Different Workload.

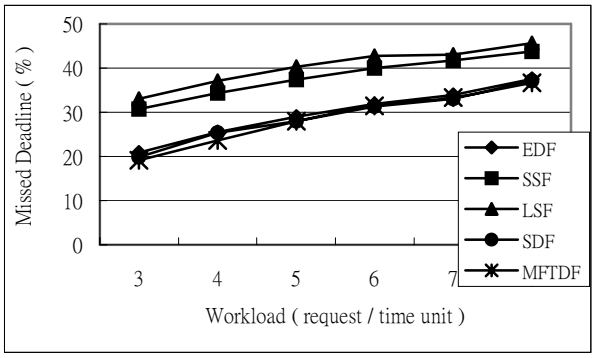


Figure 8: The Missed Deadline Over On-Demand Requests versus Different Workload.

4.4 Comparison of Different Access Pattern Model

Most of the previous researches on data broadcasting assume that access patterns of mobile clients are known and will not change with time. However, in real applications, clients may move form cell to cell. In order to model the stability of the access patterns, a parameter named *random factor* (*RF*) is introduced. RF is a value between 0 and 1. The larger the RF value is, the lower the probability of the clients following the given access pattern model will be.

In the following simulation, the effect of the changing access patterns is evaluated. The procedures of the simulation are as follows: First, a set of requests is derived as the history access pattern model. Based on this access pattern model, the broadcast program is generated. Then, a set of requests is generated according to the new access pattern model with a given RF value. In particular, RF=0 means the clients never violate the given access pattern model while RF=1 means the clients never follow the given access pattern model. We record the percentage of deadline missed versus various RF value and depict the results in the following figures.

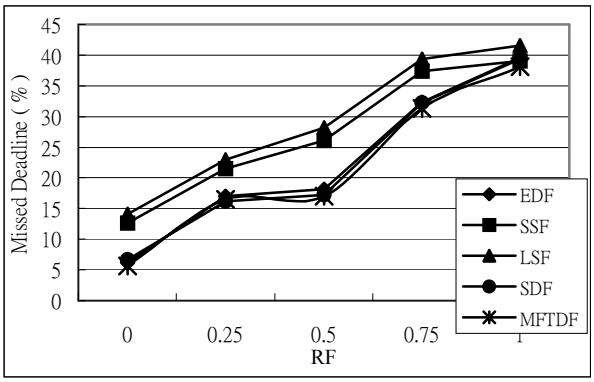


Figure 9: The Missed Deadline Over All Requests versus Different RF Value.

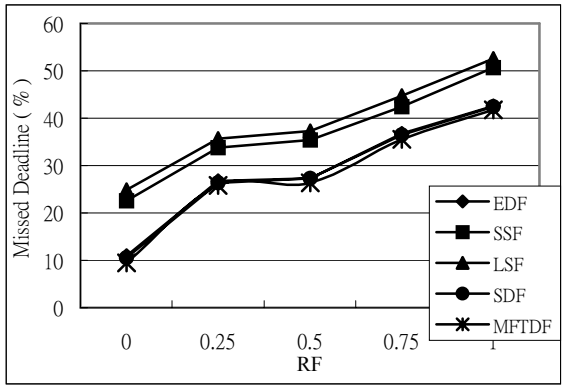


Figure 10: The Missed Deadline Over On-Demand Requests versus Different RF Value.

Figure 9and **Figure 10** illustrate the percentage of deadline missed over all requests and over on-demand requests respectively. It can be observed that the changing access patterns have a great influence on the performance of the system. For smaller RF value, more requests can be answered by the periodic broadcast program, consequently, there will be fewer requests sent to the server than those of the access model with larger RF value. As more requests sent to the server, the workload of the server increases which results in more requests miss their deadlines. Therefore, by collecting the real access patterns of the clients after a certain broadcast cycles, the broadcast scheduler can generate more effective broadcast programs.

5. CONCLUSION AND FUTURE WORK

In this paper, the problem of real-time data scheduling in multiple broadcast channels environments where the clients equip with multiple receivers is considered. The data items in the database server are classified into the broadcast data set and on-demand data set. By transforming the broadcast program generation problem from the MCMR environments to the SCSR environments, we construct the periodic broadcast program satisfying the timing

constraint. Moreover, the on-demand program generation algorithm is also proposed.

Simulation is performed to justify our approach. According to the simulation results, we can find that MFTDF, SDF or EDF have a good performance. Moreover, the time needed to insert the requests into the PQ of the proposed policy is also low.

In the future, methods of collecting the clients' access patterns can be developed. Moreover, the transmission errors can be considered in our approach.

REFERENCE:

[1] S. Jiang and N. Vaidya, "Scheduling Data Broadcast to Impatient Users", In *International Workshop on Data Engineering for Wireless and Mobile Access*, 1999.

[2] P. Xuan, S. Sen, O. Gronzalez, J. Fernandex, and K. Ramaritham, "Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments", In *Proceedings of the 3rd IEEE Symposium on Real-Time Technology and Application*, 1997.

[3] S. Baruah and A. Bestavros, "Pinwheel Scheduling for Fault Tolerant Broadcast Disks in Real-Time Database Systems", In *Proceedings of the 13th International Conference on Data Engineering*, pages 543~551, 1997.

[4] K. L. Tan, and J. Cai, "Batch Scheduling for Demand-Driven Servers in Wireless Environments", In *Information Sciences*, pages 209~231, 1998.

[5] D. Aksoy, and M. Franklin, "Scheduling for Large-Scale On-Demand Data Broadcasting", In *Proceedings of the 1998 IEEE INFOCOM Conference*, 1998.

[6] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Data On Air: Organization and Access", In *IEEE Transactions on Knowledge and Data Engineering*, 353~372, 1997.

[7] S. Acharya, M. Franklin, and S. Zdonik, "Balancing Push and Pull for Data Broadcast", In *Proceedings of the 1997 ACM-SIGMOD International Conference on Management of Data*, pages 183~194, 1997.

[8] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments", In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 199~210, 1995.

[9] S. Acharya, R. Alonso and S. Zdonik, "Dissemination-based Data Delivery Using Broadcast Disks", In *IEEE Personal Communications*, 1995.

[10] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel, "The Pinwheel: A Real-Time Scheduling Problem", In *Proceedings of 22nd International Conference on System Science*, pages 693~702, 1989.

[11] S. Baruah, and S. Lin, “Improved Scheduling of Generalized Pinwheel Task Systems”, In *Proceedings of 4th International Workshop on Real-Time Computer Systems Applications*, 1997.

[12] S. Baruah, N. Cohen, G. Plaxton, and D. Varvel, “Proportionate Progress: A Notion of Fairness in Resource Allocation”, In *Algorithmica*, pages 600~625, 1996.

[13] S. Baruah, Member IEEE, and Shun-Shii Lin, “Pfair Scheduling of Generalized Pinwheel Task Systems”, In *IEEE Transactions on Computers*, pages 812~816, 1998.

[14] S. C. Lo, and Arbee L. P. Chen, “Optimal Index and Data Allocation in Multiple Broadcast Channels”, In *IEEE International Conference on Data Engineering*, pages 293~302, 2000.

[15] S. Hameed, and N. H. Vaidya, “Efficient Algorithms for Scheduling Data Broadcast”, In ACM-Baltzer Wireless Networks, pages 183~193, 1999.

[16] K. L. Tan, and B. C. Ooi, “Data Dissemination in Wireless Computing Environments”, *Kluwer Academic Publishers*, 2000.

[17] J. Fernandex, and K. Ramaritham, “Adaptive Dissemination of Data in Time-Critical Asymmetric Communication Environments”, In *EuroMicro Conference on Real-Time Systems*, 1997.

[18] N. Shivakumar and S. Venkatasubramanian, "Energy-Efficient Indexing For Information Dissemination In Wireless Systems," In *ACM, Journal of Wireless and Nomadic Application*, 1996.

[19] Chih-Hao Hsu, Guanling Lee and Arbee L.P. Chen, “Index and Data Allocation on Multiple Broadcast Channels Considering Data Access Frequencies,” In 3rd International Conference on Mobile Data Management (MDM'2002).

[20] Chih-Hao Hsu, Guanling Lee and Arbee L.P. Chen, “A Near Optimal Algorithm for Generating Broadcast Programs on Multiple Channels,” In the 10th International Conference on Information and Knowledge Management (CIKM 2001).