Mobile Networks and Applications 8, 355–364, 2003

Dynamic Leveling: Adaptive Data Broadcasting in a Mobile Computing Environment

WEN-CHIH PENG, JIUN-LONG HUANG and MING-SYAN CHEN *
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, ROC

Abstract. The research issue of broadcasting has attracted a considerable amount of attention in a mobile computing system. By utilizing broadcast channels, a server is able to continuously and repeatedly broadcast data to mobile users. From these broadcast channels, mobile users obtain the data of interest efficiently and only need to wait for the required data to be present on the broadcast channel. Given the access frequencies of data items, one can design proper data allocation in the broadcast channels to reduce the average expected delay of data items. In practice, the data access frequencies may vary with time. We explore in this paper the problem of adjusting broadcast programs to effectively respond to the changes of data access frequencies, and develop an efficient algorithm DL to address this problem. Performance of algorithm DL is analyzed and a system simulator is developed to validate our results. Sensitivity analysis on several parameters, including the number of data items, the number of broadcast disks, and the variation of access frequencies, is conducted. It is shown by our results that the broadcast programs adjusted by algorithm DL are of very high quality and are in fact very close to the optimal ones.

Keywords: broadcast disks, mobile computing, broadcast programs, multiple broadcast channels

1. Introduction

In a mobile computing environment, a mobile user with a power-limited mobile computer can access various information via wireless communication. Applications such as stock activities, traffic reports and weather forecast have become increasingly popular in recent years [21,22]. It is noted that mobile computers use small batteries for their operations without directly connecting to any power source, and the bandwidth of wireless communication is in general limited. As a result, an important design issue in a mobile system is to conserve the energy and communication bandwidth of a mobile unit while allowing mobile users of the ability to access information from anywhere at anytime [2,4,10].

The research issue of broadcasting has attracted a considerable amount of attention in a mobile computing system. By utilizing broadcast channels, a server is able to continuously and repeatedly broadcast data to mobile users. These broadcast channels are also known as "broadcast disks" from which mobile users can obtain the data of interest efficiently and only need to wait for the required data to present on the broadcast channel [1,9,20]. The corresponding waiting time is called the expected delay of that data item. One objective of designing proper data allocation in the broadcast disks is to reduce the average expected delay of data items. Broadcasting schemes in this context have been extensively studied [3,8,12,15,18].

Note that a lot of research effort has been elaborated on exploring multiple broadcast channels for data dissemination [13,14,16,17]. The advantages of utilizing multiple broadcast channels can be found in [16,17]. Organizing data in multiple broadcast channels raises a number of new research problems. A system of multiple broadcast channels can be viewed as a *broadcast disk array.* The broadcast disks in a broadcast disk array can be categorized according to the *speed* of broadcast disks, where the speed of a broadcast disk corresponds to the expected delay for the data items in that broadcast disk. The data items in each broadcast disk are sent out in a round robin manner. Clearly, as the number of data items in a broadcast disk increases, the expected delay of those data items increases. As a result, the data items that are more frequently requested by mobile users should be put in fast broadcast disks, whereas cold data items can be pushed to slow broadcast disks to minimize the average expected delay of data items in the broadcast disk array. Thus, it has been recognized as an important issue to develop algorithms to allocate data items to the broadcast disk array according to their access frequencies so as to minimize the average expected delay of data items [14,16].

The study in [14] explored the problem of generating hierarchical broadcast programs with the data access frequencies and the number of broadcast disks in a broadcast disk array given. Specifically, the problem of generating hierarchical broadcast programs is first transformed into the one of constructing a channel allocation tree with variant-fanout. By exploiting the feature of tree generation with variant-fanout, a heuristic algorithm to minimize the expected delay of data items in the broadcast program is developed. The tree obtained in [14] is called *channel allocation tree* (or abbreviatedly as allocation tree) where the depth of the allocation trees corresponds to the number of broadcast disks, and those leaf nodes in the same level of the allocation tree correspond to those data items to be put in the same broadcast disk. Figure 1 shows a hierarchical broadcast program with its channel allocation tree where the upper channel is allocated with two data items and each of the three lower channels is allocated

* Corresponding author.

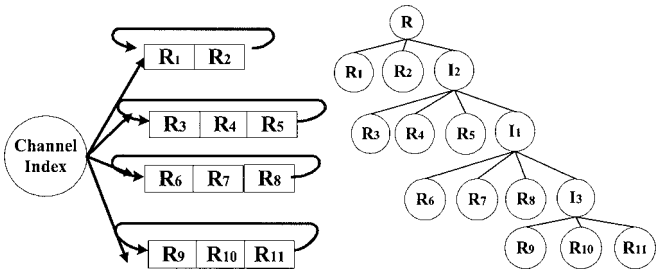

Figure 1. The broadcast program and its allocation tree.

with three data items. As such, the data items in the fast disks (i.e., the upper broadcast channel) spin faster than those data items in the slow disks (i.e., the lower broadcast channels). Note, however, that the algorithm in [14] is designed for the situation where the data access frequencies and the number of broadcast channels are given. In practice, the data access frequencies may vary as time advances. For example, the access frequencies of the traffic data increase drastically during rush hours and decrease beyond the rush hours. Clearly, without adapting to the change of access frequencies, the broadcast program determined off-line will unavoidably lead to degraded performance. Thus, with the broadcast programs generated by [14], it is important for the broadcast programs to dynamically adapt to the change of the data access frequencies so as to retain the performance of data broadcasting. This is the very problem that we shall address in this paper.

The problem we study can be best understood by the illustrative example in table 1. Assume that the data items R_i, $1 \leqslant i \leqslant 11$ are of the same size and the number of broadcast channels is 4. Denote that the access frequency of data item R_i as $P_r(R_i)$. Four sets of access frequencies of data items are given in table 1 and drawn in figure 2 for clarity.[1] The average expected delay in table 1 is obtained by multiplying the access frequency of each data item by the expected delay of that data item and summing up the results, i.e., $\sum_{i=1}^{11} d_{R_i} \cdot P_r(R_i)$. Same as in [1,19], the expected delay for each data item in the broadcast disk i is formulated as $\sum_{x=1}^{N_i}(N_i - x)/N_i$, where N_i is the number of data items allocated in the broadcast disk i. It can be verified that the expected delays of data items R_1, R_3, R_5 and R_9 in figure 1 are $d_{R_1} = (1 + 0)/2 = 0.5$, $d_{R_3} = (2 + 1 + 0)/3$, $d_{R_6} = (2 + 1 + 0)/3$ and $d_{R_9} = (2 + 1 + 0)/3$, respectively.

At time t_0, with the access frequencies of data items and the number of broadcast channels given, the initial allocation trees obtained by the work in [14] is shown in figure 1. The average expected delay of data items at time t_0 is $\sum_{i=1}^{11} d_{R_i} \cdot P_r(R_i) = 0.8712$. Assume that the allocation tree will remain the same as the access frequencies of data items vary with time. With the allocation tree determined at time t_0, the average expected delay of data items at time t_4 is $\sum_{i=1}^{11} d_{R_i} \cdot P_r(R_i) = 0.7371$. Notice that with the access frequencies changed, this average expected delay at time t_4 is much larger than its optimal value[2] (which is 0.5557 in this case). Thus, it is an important issue to dynamically adjust the broadcast program to reflect the change of access frequencies. Consequently, date items should be moved among levels of a given allocation tree to adapt to the change of access frequencies of data items.

In this paper, by shuffling data items among different levels in the allocation tree, we devise an algorithm to dynamically adjust the broadcast programs in response to the change of data access frequencies. This algorithm is referred to as algorithm DL (standing for dynamic leveling). Clearly, a naive approach to reach a new configuration would be re-executing the algorithm in [14] again, which is however costly. Algorithm DL is so designed that the new configuration for efficient broadcast programs can be reached with the purpose of minimizing the number of data movements. Notice that once the change of access frequencies is larger than the predetermined value (Such a value is called *fluctuation factor),* algorithm DL should be executed to dynamically adjust broadcast programs. Explicitly, the process of algorithm DL can be decomposed into two phases, namely (1) the casual adjustment phase and (2) the fine adjustment phase. In the casual adjustment phase, algorithm DL reaches an initial adjustment for data items among broadcast channels. Then, for fine tuning, algorithm DL is designed to adjust the data items between neighboring levels in the fine adjustment phase with the objective of minimizing the total cost of these two neighboring levels. Performance of algorithm DL is analyzed and a system simulator is developed to validate our results. Sensitivity analysis on several parameters, including the number of data items, the number of broadcast disks, and the variation of access frequencies, is conducted. It is shown by our simulation results that the broadcast programs achieved by algorithm DL are of very high quality and are in fact very close to the optimal ones. This feature and the efficiency of algorithm DL justify the practical importance of algorithm DL.

The rest of this paper is organized as follows. Problem description is given in section 2. In section 3, we develop algorithm DL to adjust the allocation tree to reflect the change of access frequencies. Performance studies are conducted in section 4. This paper concludes with section 5.

2. Problem description

Table 2 shows the descriptions of symbols used in this paper. Denote the total number of data items as n, and a data item as R_i, $1 \leqslant i \leqslant n$. The number of broadcast disks in a broadcast disk array is K. Recall that $P_r(R_i)$ is the access frequency of R_i and $\sum_{i=1}^{n} P_r(R_i) = 1$. Theoretically, generating a broadcast program can be viewed as a partition problem for data items. Given the number of broadcast disks in a disk array and the access frequencies of all data items, we shall determine the proper set of data items that should be allocated to each broadcast disk in a broadcast disk array with the purpose

[1] The access frequencies of data items are generated by Zipf distribution which will be described later in this paper.

[2] Such optimal values can be obtained by exhaustive searches for broadcast programs, and will be used for comparison purposes in this paper.

Table 1
Access frequencies of data items.

Time	Access frequency											Average expected delay	
	$P_r(R_1)$	$P_r(R_2)$	$P_r(R_3)$	$P_r(R_4)$	$P_r(R_5)$	$P_r(R_6)$	$P_r(R_7)$	$P_r(R_8)$	$P_r(R_9)$	$P_r(R_{10})$	$P_r(R_{11})$	in figure 1	optimal
t_1	0.126	0.123	0.116	0.11	0.10	0.095	0.0869	0.0777	0.0673	0.055	0.0388	0.8712	0.8712
t_2	0.16	0.152	0.137	0.122	0.10	0.09	0.0763	0.061	0.0458	0.0305	0.0152	0.8338	0.7805
t_3	0.2226	0.201	0.163	0.129	0.09	0.07	0.0504	0.0323	0.0181	0.008	0.002	0.7746	0.7148
t_4	0.276	0.239	0.194	0.102	0.08	0.05	0.0298	0.0153	0.0064	0.0019	0.0002	0.7371	0.5557

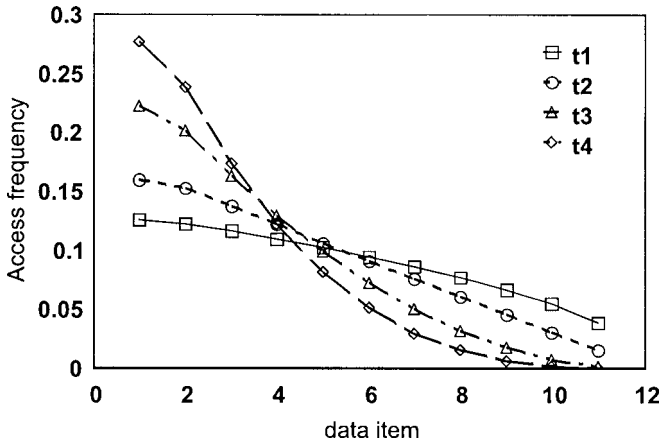

Figure 2. Data access frequencies vary as time advances.

Table 2
Description of symbols.

Description	Symbol
Number of broadcast disks in a broadcast disk array	K
Number of data items within broadcast disk i	N_i
The expected delay of data items within broadcast disk i	d_i
The jth data item	R_j
The access frequency of data item R_j	$P_r(R_j)$

of minimizing the average expected delay of all data items (i.e., $\sum_{i=1}^{n} d_{R_i} \cdot P_r(R_i)$). The problem of generating broadcast programs for K broadcast channels can be viewed as a discrete minimization problem: Given a list of n data items with their access probabilities, partition them into K parts so that the average expected delay of all data items is minimized. The minimization problem is known to be NP-hard [11]. As pointed out in [14], a broadcast program for a broadcast array of K broadcast disks can be represented as a channel allocation tree with a height of K. Note that the leaf nodes in the same level of the allocation tree correspond to a set of data items to be put in the same broadcast disk.

To facilitate the presentation of the costs for an allocation tree, we have the following definition.

Definition 1. Suppose that level v in the allocation tree has $j - i + 1$ data items, $R_i, R_{i+1}, \ldots, R_j$. The cost of level v in an allocation tree is defined as $C_{i,j} = \sum_{k=1}^{j-i+1}((j - i + 1) - k)/(j - i + 1) \sum_{q=i}^{j} P_r(R_q)$. In essence, the value of C_{ij} is related to the average expected delay of leaf nodes in level v.

This paper investigates the problem of adjusting the broadcast program to match the access frequencies of data items. In order not to distract readers from the main theme of this paper for dynamically adjusting broadcast programs, readers interested in the details of collecting access frequencies are referred to [6,7,18,23]. Once the change of access frequencies is larger than the predetermined value, algorithm DL will be executed to reach the new configuration close to the optimal one. Figure 3 shows the optimal allocation trees with the access frequencies given in table 1. In accordance with the access frequencies of data items at time t_1 and the number of broadcast channels given, the allocation tree was determined by the algorithm in [14]. It can be seen in figure 3, at time t_2, t_3 and t_4, the optimal allocation trees differ from the one at time t_1 due to the change of access frequencies. Consequently, date items should be moved among levels within the given allocation tree in response to the change of access frequencies of data items. Clearly, such movements have an impact on the average expected delay of all data items. The problem we shall study in this paper can be stated as follows.

Problem of adjusting allocation trees. Given an allocation tree, we shall adjust data items among the broadcast disks when the access frequencies vary with the purpose of minimizing the expected delay of data items.

With the problem described above, we should devise an algorithm to determine the level of the allocation tree to start the adjustment and identify the movements of data items among levels in the allocation tree.

3. Algorithm DL: adjusting allocation tree by dynamic leveling

In section 3.1, we devise algorithm DL to adjust allocation trees which explores the features of the casual adjustment and the fine adjustment. Then, the execution scenario of algorithm DL is illustrated in section 3.2.

3.1. Design of algorithm DL

We devise in this paper an algorithm, referred to as algorithm DL, to dynamically adjust the broadcast programs by shuffling data items among different levels in the allocation tree. The process of algorithm DL can be decomposed into two phases, namely (1) the casual adjustment phase and (2) the fine adjustment phase. In the casual adjustment phase, algorithm DL moves data items among levels so as to enable the

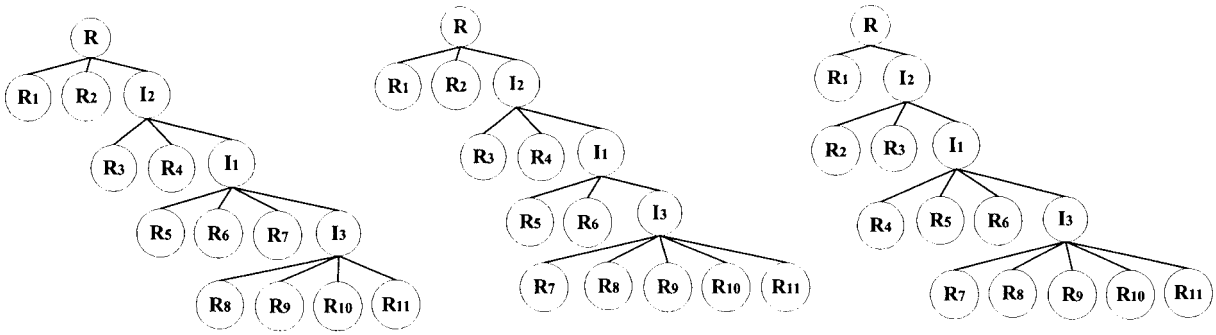

Figure 3. The optimal allocation trees under different times.

costs of most levels in the allocation tree to be smaller than or equal to average cost. Then, for fine tuning, algorithm DL adjusts the data items between neighboring levels with the objective of minimizing the total cost of these two neighboring levels. Note that algorithm DL is greedy in nature and is of time complexity $O(K + n)$. The algorithmic form of algorithm DL is described below.

Algorithm DL.
Input: The status table (ST) with K rows, where K is the number of broadcast disks in a broadcast disk array.
Output: The resulting allocation tree.
begin
1. **for** each row i in table ST
2. $ST(i) \cdot D = ST(i) \cdot C - ST(i) \cdot P$;
3. $ST(i) \cdot G =$ false; /* $ST(i) \cdot G$ is the flag for casual checking*/
/*Array δ has $K - 1$ elements which record the cost difference between two neighboring level*/
4. **for** each element i in array δ
5. $\delta[i] = |ST(i) \cdot C - ST(i + 1) \cdot C|$
6. casual_checking $= 0$;
/*The casual adjustment phase*/
7. Choose the row i from table ST such that $ST(i) \cdot D$ is maximal
8. **repeat**
9. **begin**
10. **if** $(i == 1)$
11. casual_tunning$(i, i + 1)$;
12. **else if** $(i == k)$
13. casual_tunning$(i, i - 1)$;
14. **else**
15. {choose the row j where $j \in (i - 1, i + 1)$ such that $ST(j) \cdot G$ is false and $\delta[j]$ is maximal;
16. casual_tunning(i, j);}
17. casual_checking++;
18. update table ST and array δ accordingly;
19. choose the row i from ST where $ST(i) \cdot C$ is maximal and $ST(i) \cdot G$ is false;
20. **end**
21. **until** casual_checking $== K - 1$;
/*the fine adjustment phase*/
22. Construct a priority queue PQ;
/*A priority queue is a data structure which returns the element with the minimal value when one is to remove an element from the priority queue*/
23. **for** each element i in array δ
24. Insert $\delta[i]$ into the PQ;
25. **while** (PQ is not empty)
26. **begin**
27. remove the element i from PQ;
28. **if** $(ST(i) \cdot C < ST(i + 1) \cdot C)$
/*if there is no movement between level i and level $i + 1$, moving equals to -1*/
29. moving=push_up$(i, i + 1)$;
30. **else**
31. moving=push_down$(i, i + 1)$;
32. **if** (moving$= -1$) /*some data movements occur*/
33. Update the elements in PQ and table ST accordingly;
34. **end**
end

Procedure casual_tuning(*level i*, *level j*)
{
sort those data items in level i according their access probabilities;
if $(i < j)$
begin
while $(ST(i) \cdot C > \sum_{i=1}^{K} ST(i) \cdot C/K)$
move the data item in the rightest side of level i to level j and update $ST(i) \cdot C$ accordingly;
end
else
begin
while $(ST(i) \cdot C > \sum_{i=1}^{K} ST(i) \cdot C/K)$
move the data item in the leftest side of level i to level j and update $ST(i) \cdot C$ accordingly;
end
}

Table ST (standing for status table) is created to record the cost of each level in the allocation tree, and the number of rows in table ST is equal to the number of broadcast disks in a broadcast disk array (from line 1 to line 3). Note that in

table ST, the value of $ST(i) \cdot P$ is the cost of nodes in level i previously, whereas the value of $ST(i) \cdot C$ is the cost of nodes in level i when the latest access frequencies were collected. $ST(i) \cdot D$ stores the cost difference associated with level i, i.e., $ST(i) \cdot C - ST(i) \cdot P$. Also, $ST(i) \cdot G$ is used to indicate whether the casual tuning is performed or not. Array δ has $K - 1$ elements that record the cost difference between two neighboring levels (from line 4 to line 5). As can be seen in causal adjustment phase (i.e., from line 7 to line 21), algorithm DL makes sure that most levels of the allocation tree satisfy the requirement of the casual adjustment. Since the casual adjustment intends to let the total cost of allocation tree be evenly allocated to all levels, it is possible that some data nodes would move back and forth between neighboring levels. For execution efficiency, the number of runs for the casual adjustment is limited to be $K - 1$. Procedure casual_tuning is developed to move data items in level i so as to satisfy the purpose of the casual adjustment.

By exploiting the casual adjustment, data items are roughly allocated to each level of an allocation tree with the costs of most levels are smaller than or equal to average cost. Then, algorithm DL employs the fine adjustment to adjust data items between neighboring levels. As can be seen from line 22 to line 34 of algorithm DL, neighboring levels are examined on finding potential movements with the purpose of minimizing the total cost of neighboring levels. Specifically, in line 27 of algorithm DL, the sequence of performing the fine tuning is determined by identifying the largest cost difference among those between neighboring levels (i.e., the largest value in δ). After identifying the neighboring levels (e.g., level i and level $i + 1$) to perform the fine tuning, one should determine the data movements between these levels. Note that there are two kinds of movements, i.e., pushing up and polling down. Judiciously applying these movements is able to reduce the total cost of these two neighboring levels. Clearly, if the cost of level i is smaller than level $i + 1$, we should move data items from level $i + 1$ to level i and vice versa. After deciding the direction of data movements, we should determine the number of data items to move among levels in an allocation tree. Explicitly, we develope procedure push_up and pull_down to determine such a number. To facilitate the presentation of algorithm DL, the procedures of push_up and pull_down are described in detail later. From line 26 to line 34, algorithm DL adjusts data items in neighboring levels iteratively with the objective of minimizing the total cost of neighboring levels until there is no further adjustment required (i.e., queue PQ is empty). As such, the allocation tree is adjusted so as to minimize the total cost of the allocation tree.

Once we identify the direction of data movements to perform, we should determine the number of data items to move among levels. Suppose that data items in each level of an allocation tree are sorted according to the descending order of access frequencies. In order to evaluate the cost reduction by the movement of pushing up, we have the following definition.

Definition 2. Suppose that level i has $k - i + 1$ nodes, $R_i, R_{i+1}, \ldots, R_k$ and level $i + 1$ has $j - k$ data nodes, $R_{k+1}, R_{k+2}, \ldots, R_j$. The reduction gain achieved by pushing p nodes (i.e., $R_{k+1}, R_{k+2}, \ldots, R_{k+p}$) up to level i, denoted by $u(p)$, can be formulated as $u(p) = (C_{i,k} + C_{k+1,j}) - (C_{i,k+p} + C_{k+p+1,j})$.

In light of definition 2, we devise a procedure push_up to identify the group of nodes in level $i + 1$ to be moved upward to level i so as to maximize the reduction gain between these two neighboring levels. Figure 4 shows the scenario of pushing up.

Procedure push_up(*level i*, *level i* + 1)
{
 Determine p^* such that $u(p^*) = \max_{1 \leqslant p \leqslant j-k}\{u(p)\}$;
 /*determine the maximal value of $u(p)$ when p varies from 1 to $j - k$*/
 if $u(p^*) > 0$
  push nodes $R_{k+1}, R_{k+2}, \ldots, R_{k+p^*}$ to level i in the tree;
 else
  $p^* = -1$; /*no movement is performed since there is no cost-effective movement*/
}

Similar to the operation of pushing up, we have definition 3 and procedure pull_down below to evaluate the group of nodes in level i to be moved downward to level $i + 1$ with the purpose of reducing the total cost of these two neighboring levels. Figure 5 illustrates the scenario of pulling down.

Definition 3. Suppose that level i has $k - i + 1$ data nodes, $R_i, R_{i+1}, \ldots, R_k$ and level $i + 1$ has $j - k$ data nodes, $R_{k+1}, R_{k+2}, \ldots, R_j$. The reduction gain achieved by pulling p nodes (i.e., $R_{k-p+1}, R_{k-p+2}, \ldots, R_k$) down to level $i + 1$, denoted by $d(p)$, can be formulated as $d(p) = (C_{i,k} + C_{k+1,j}) - (C_{i,k-p} + C_{k-p+1,j})$.

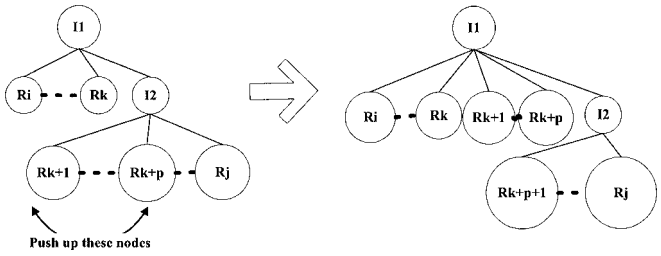

Figure 4. A scenario of pushing up.

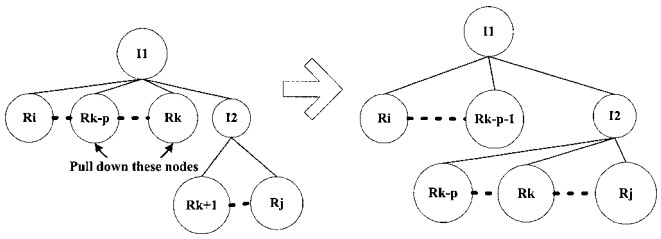

Figure 5. A scenario of pulling down.

Table 3
The profile of an illustrative example.

Time	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}
t_1	0.126	0.123	0.116	0.11	0.1	0.095	0.0869	0.0777	0.0673	0.055	0.0388
t_2	0.276	0.239	0.194	0.102	0.08	0.05	0.0298	0.0153	0.0064	0.0019	0.0002

Procedure pull_down(*level i*, *level i* + 1)
{
Determine p^* such that $d(p^*) = \max_{1 \leqslant p \leqslant k-i+1}\{d(p)\}$;
/*determine the maximal value of $d(p)$ when p varies from 1 to $k - i + 1$*/
if $d(p^*) > 0$
pull nodes $R_{k-p^*+1}, R_{k-p^*+2}, \ldots, R_k$ down level $i + 1$ in the tree;
else
$p^* = -1$; /*no movement is performed since there is no cost-effective movement*/
}

3.2. An example execution scenario of algorithm DL

Consider the profile in table 3 where the number of data items n is 11 and the number of broadcast disks is 4. The initial allocation tree is shown in figure 6(a), where the allocation tree is generated according to the access frequencies at t_1. The values in table *ST* and their changes made in accordance with the execution of algorithm DL are shown in table 4. First, algorithm DL chooses the maximal $ST(i) \cdot D$ to start the casual adjustment. In this example, level 1 is chosen since $ST(1) \cdot D$ is the largest among all levels. As can be seen in table 4(a), since the cost of level 1 is larger than the average cost of all levels (i.e., $ST(1) \cdot C = 0.2575 > \sum_{i=1}^{4} ST(i) \cdot C/4 = (0.2575 + 0.376 + 0.0951 + 0.0085)/4 = 0.184275$), data items in level 1 should be pulled down to level 2 to meet the criterion of the casual adjustment. Thus, data item R_2 is moved to level 2 and table *ST* is updated accordingly. Following the same procedure, algorithm DL performs procedure casual_tuning iteratively until casual_checking equals $K - 1$ (i.e., $4 - 1 = 3$). Table 4(b) shows the values of table *ST* and the corresponding values in array δ after the casual adjustment. The configuration of the allocation tree in figure 6(a) becomes the one shown in figure 6(b). Then, in the phase of the fine adjustment, each element of array δ with its value is inserted into queue *PQ* and algorithm DL performs the fine tuning between neighboring levels. From table 4(b), since $\delta[3]$ is the largest, the fine adjustment will be executed between level 3 and level 4. As $ST(3) \cdot C$ is smaller than $ST(4) \cdot C$, those data items in level 4 should be pushed up to level 3. It can be verified that data item R_5 should be pushed up to level 3 so as to reduce the total cost of level 3 and level 4. Table 4(c) shows the values of table *ST* and the corresponding values in array δ after the fine tuning is performed between level 3 and level 4. Figure 6(c) shows the configuration of the allocation tree after the first fine tuning. From table 4(c), the next fine tuning is performed between level 2 and level 3. Following the fine adjustment, algorithm DL stops when there is no further data movement required. Table 4(d) shows the values of table *ST* after performing algorithm DL. The final allocation tree is shown in figure 6(d). Note that in this case the final allocation tree happens to be the optimal one shown in figure 3(c).

Table 4
An execution scenario under algorithm DL. (a) Run 1 of algorithm DL, (b) table *ST* after the phase of the casual adjustment, (c) table *ST* after the first fine tuning is performed, (d) the final result of table *ST*.

(a)

Level i	$ST(i) \cdot P$	$ST(i) \cdot C$	$ST(i) \cdot D$	$ST(i) \cdot G$
1	0.1245	0.2575	0.133*	false
2	0.326	0.376	0.05	false
3	0.25	0.0951	−0.1549	false
4	0.1611	0.0085	−0.1526	false

(b)

Level i	$ST(i) \cdot P$	$ST(i) \cdot C$	$ST(i) \cdot D$	$ST(i) \cdot G$	δ
1	0.1245	0	−0.1245	true	0
2	0.326	0	−0.326	true	0.148
3	0.25	0.148	−0.102	true	0.4028*
4	0.1611	0.5508	0.3897	false	

(c)

Level i	$ST(i) \cdot P$	$ST(i) \cdot C$	$ST(i) \cdot D$	$ST(i) \cdot G$	δ
1	0.1245	0	−0.1245	true	0
2	0.326	0	−0.326	true	0.376*
3	0.25	0.376	0.126	true	0.117
4	0.1611	0.259	0.0979	false	

(d)

Level i	$ST(i) \cdot P$	$ST(i) \cdot C$	$ST(i) \cdot D$	$ST(i) \cdot G$
1	0.1245	0	−0.1245	true
2	0.326	0.2165	−0.1095	true
3	0.25	0.252	0.002	true
4	0.1611	0.1072	−0.0539	false

4. Performance evaluation

In order to evaluate the performance of algorithm DL, we have implemented a simulation model of the broadcast environment. Specifically, the simulation model is described in section 4.1. Then, we examine the impact of adjusting broadcast programs in section 4.2. Performance of algorithm DL is analyzed in section 4.3. In section 4.4, algorithm DL and the work in [14] is comparatively analyzed.

4.1. Simulation model

Table 5 summarizes the definitions for some primary simulation parameters. The number of data items to be broadcasted

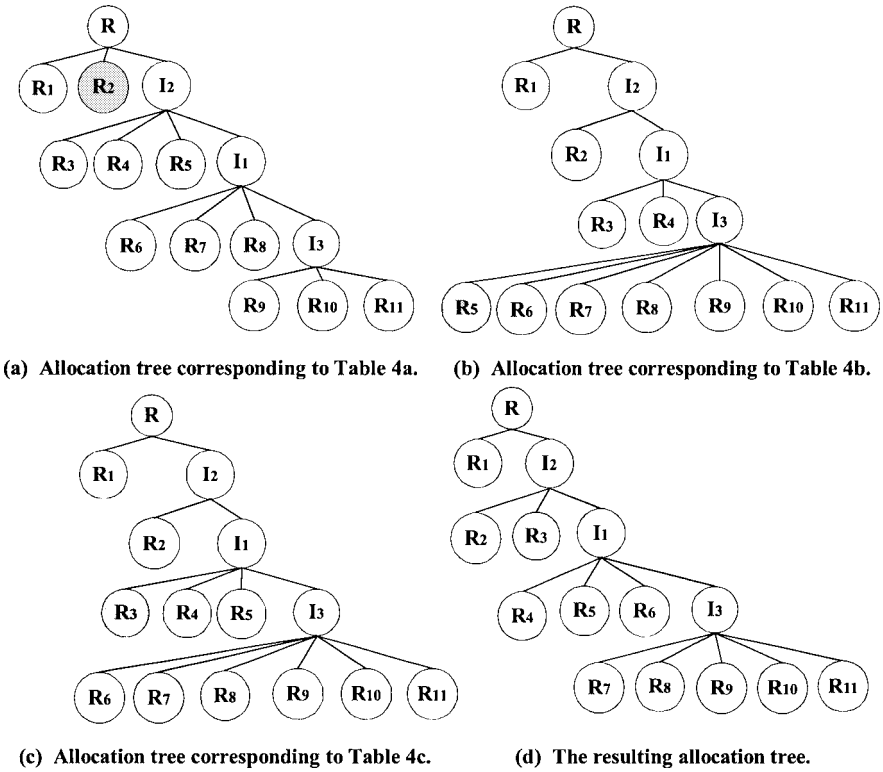

Figure 6. An execution scenario of algorithm DL: (a)–(c) the adjustment of the allocation tree, and (d) the resulting allocation tree.

Table 5
The parameters used in the simulation.

Notation	Definition
n	total number of data items to be broadcast
K	number of broadcast disks in a broadcast disk array
θ	Zipf parameter
f	fluctuation factor
no_adjust	scheme which does not adjust the broadcast program
OPT	scheme to generate the optimal broadcast program
VF^K	scheme to generate the broadcast programs statically

in a broadcast disk array is denoted by n and the number of broadcast disks in a broadcast disk array is K. The access frequencies of broadcast data items are modelled by the Zipf distribution. Let $P_r(R_i) = ((N-i)/N)^\theta / \sum_{j=1}^{n}((N-j)/N)^\theta$, where θ is the parameter of Zipf distribution [5]. It can be verified that the access frequencies become increasingly skewed as the value of θ increases. Specifically, the initial Zipf parameter, denoted by θ_0, is set to 0.5. θ_{current} is the Zipf parameter for the current access frequencies, whereas θ_{previous} is the Zipf parameter collected last. The number of f called the fluctuation factor is used to determine whether algorithm DL will be executed or not. If the difference between θ_{current} and θ_{previous} is larger than the value of f, DL will be executed to adjust the broadcast program in order to retain the performance. For comparison purposes, a scheme, no_adjust, which does not adjust the broadcast program in response to the change of access frequencies, is implemented. To obtain the optimal solutions for comparison, we implemented scheme OPT by using the technique of branch and bound [11]. For interest of brevity, the implementation details of OPT are omitted in this paper. Notice that though scheme OPT is able to find the optimal broadcast program, the execution time of OPT is prohibitively large due to its exponential time complexity. For comparison purposes, we also implemented scheme VF^K, which is able to generate broadcast program with the number of broadcast disks and the number of data items given.

4.2. The impact of adjusting broadcast programs

To show the advantage of adjusting broadcast programs when the access frequencies vary, we set the value of n to 50, the value of K to 4 and the value of f to 1. The expected delays of data items under no_adjust, DL and OPT are examined with the value of θ varied. Without loss of generality, assume that all the data items are of the same size which is used as one unit of waiting time. The initial broadcast program is generated by scheme OPT. The resulting expected delays of data items by running no_adjust, DL and OPT are shown in figure 7. It can be seen from figure 7 that the access frequencies become increasingly skewed as the value of θ increases and the average expected delay of DL decreases since the broadcast program is properly adjusted by DL in accordance with the change of access frequencies of data items. Note that the difference between expected delay of DL and that of no_adjust becomes larger as the θ increases, indicating the necessity of adjusting broadcast programs while the access frequencies of data items vary. It is worth mentioning that though algorithm DL is applied in 5 times, the expected delays of DL and OPT are still very close in figure 7, showing the good quality of configurations adjusted by algorithm DL.

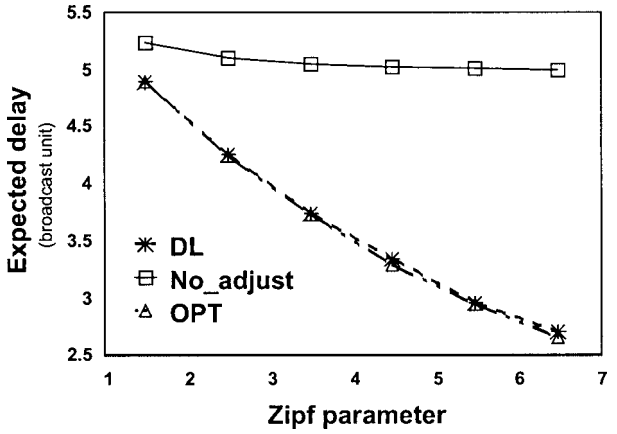

Figure 7. The average expected delays of no_adjust, DL and OPT with the value of Zipf parameter varied.

4.3. The performance of algorithm DL

We now investigate the quality of solutions obtained by DL and OPT. Note that algorithm DL will dynamically adjust the broadcast program while the Zipf parameter varies. In order to evaluate the impact of increasing the value of f, we set the value of n to 50 and the value of K to 4. Figure 8 shows the performance results of OPT and DL. As can be seen in figure 8, the difference between expected delay of DL and that of OPT is almost negligible, showing the very high quality of the solutions obtained by algorithm DL. Note that as the value of f increases, the solutions obtained by algorithm DL are all very close to the optimal ones, indicating the robustness in algorithm DL.

Next, the experiments of varying the value of K for OPT and DL are conducted where we set the value of n to be 50 and the value of f to be 2.5. Figure 9 shows the average expected delays of OPT and DL with the value of K varied. As the value of K increases, the expected delays of OPT and DL decrease. This agrees with our intuition since as the number of broadcast channels increases, the number of data items in each broadcast channel decreases, thereby reducing the expected delay of data items. Notice that the difference between the expected delay of DL and that of OPT is very small, again showing the good quality of solutions obtained by DL. The performance of DL with the value of n varied is examined where we set the value of K to 5 and the value of f to 2.5. The average expected delays of OPT and DL with the value of n varied are shown in figure 10. As the number of data items to be broadcast increases, the expected delays of data items resulted by OPT and DL increase linearly as we anticipate. Also, the difference between the expected delays resulted by OPT and DL is negligible.

4.4. Comparative analysis for VF^K and DL

In [14], VF^K is designed for the situation where the data access frequencies and the number of broadcast channels are given. The experimental results show that the broadcast program generated by VF^K is of very high quality. However,

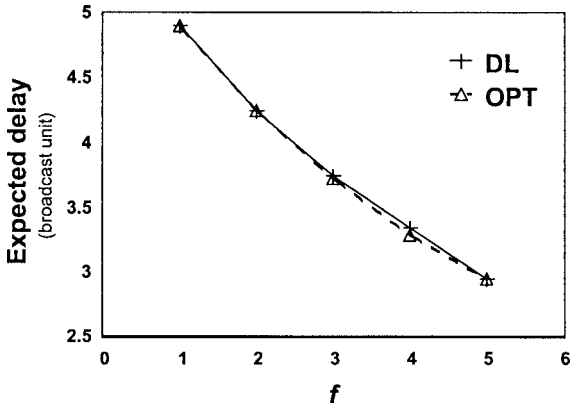

Figure 8. The average expected delays of OPT and DL with the value of f varied.

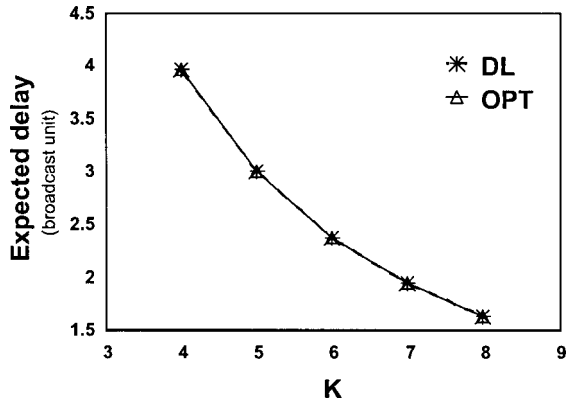

Figure 9. The average expected delays of OPT and DL with the value of K varied.

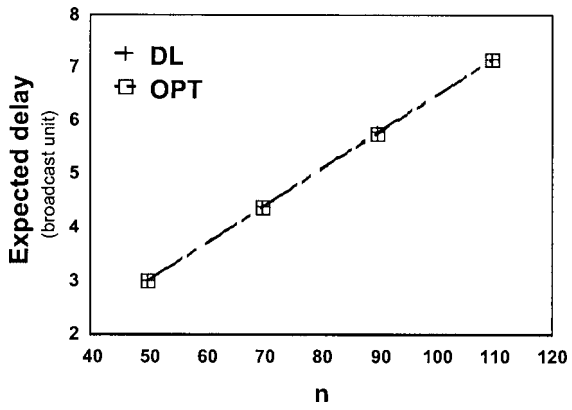

Figure 10. The average expected delays of OPT and DL with the value of n varied.

in practice, the data access frequencies may vary as time advances. It is important for broadcast programs to adapt to the change of the data access frequencies so as to retain the performance of data broadcasting. In this section, our experimental results show that algorithm DL is more efficient to achieve new configuration without re-executing VF^K.

To evaluate the impact of increasing the value of n, we set the value of K to 15 and the value of f to one. Figure 11 shows the execution times incurred by VF and DL. In fig-

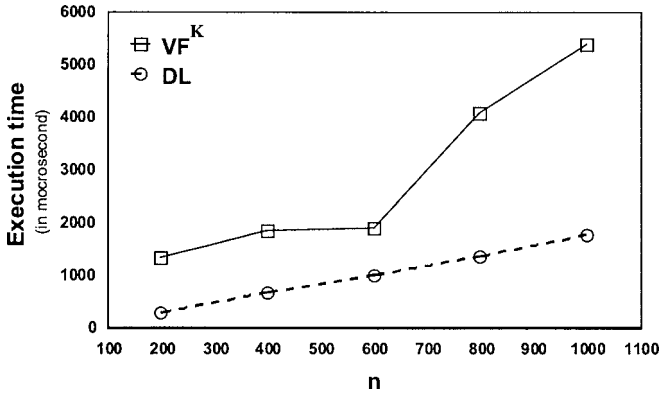

Figure 11. The execution times incurred by VF^K and DL with the value of n varied.

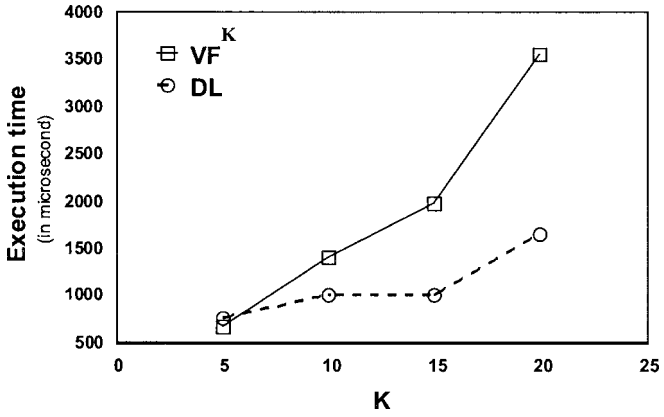

Figure 12. The execution times incurred by VF^K and DL with the value of K varied.

ure 11, the execution times incurred by VF^K and by DL increase as the number of data items increases. Note that the execution time incurred by VF^K is larger than that incurred by DL, showing that DL is able to achieve the new configuration more efficiently when the data access frequencies vary. It is also observed that the curve of VF^K in figure 11 is not as smooth as that of DL due to the lack of dynamic allocation adjustment of VF^K.

Next, we examine the impact of increasing the value of K. Without loss of generality, we set the value of n to 5 and the value of f to one. The execution times incurred by VF^K and DL with the value of K varied are shown in figure 12. Notice that when the value of K is smaller than 5, the execution time incurred by VF^K is smaller than that incurred by DL. However, as the value of K increases, the execution time incurred by VF^K is significantly larger than that incurred by DL. This indicates that when the value of K increases, the advantage of algorithm DL over the approach of re-executing algorithm VF^K increases. In all, algorithm DL is able not only to adjust the broadcast program efficiently in response to the change of access frequencies of data items but also to produce the solutions of very high quality. Note that the capability of adjusting broadcast programs dynamically should be viewed as an enhanced feature rather than a limitation. In fact, when the value of f is set to be infinite, the initial broadcast program generated by VF^K will not be adjusted according to the change of access frequencies. Thus, the performance degrades as the access frequencies vary, justifying the necessity of algorithm DL.

5. Conclusions

We explored in this paper the problem of adjusting broadcast programs to cope with the data access frequencies varied. By exploiting the features of the casual adjustment and the fine adjustment, we developed a heuristic algorithm DL to adjust broadcast programs when the access frequencies of data items change. Performance of algorithm DL was analyzed and a system simulator was developed to validate our results. Sensitivity analysis on several parameters, including the number of data items, the number of broadcast disks, and the variation of access frequencies, was conducted. It was shown by our simulation results that the broadcast programs achieved by algorithm DL are of very high quality and are in fact very close to the optimal ones. This feature and the efficiency of algorithm DL justify the practical importance of algorithm DL.

References

[1] S. Acharya, R. Alonso, M. Franklin and S. Zdonik, Broadcast disks: Data management for asymmetric communication environments, in: *Proceedings of ACM SIGMOD* (March 1995) pp. 199–210.

[2] D. Barbara, Mobile computing and databases – a survey, IEEE Transactions on Knowledge and Data Engineering 11(1) (January/February 1999) 108–117.

[3] M.-S. Chen, P.S. Yu and K.-L. Wu, Indexed sequential data broadcasting in wireless mobile computing, in: *17th IEEE International Conference on Distributed Computing Systems* (1997) pp. 124–131.

[4] M.H. Dunham, Mobile computing and databases, *Tutorial of International Conference on Data Engineering* (February 1998).

[5] J. Gray, P. Sundaresan, S. Englert, K. Baclawski and P. J. Weinberger, Quickly generating billion-record synthetic databases, in: *Proceedings of ACM SIGMOD* (March 1994) pp. 243–252.

[6] Q.L. Hu, D.L. Lee and W.-C. Lee, Dynamic data delivery in wireless communication environments, in: *Proceedings of International Workshop on Mobile Data Access* (November 1998) pp. 218–229.

[7] Q. Hu, D.L. Lee and W.-C. Lee, Performance evaluation of a wireless hierarchical data dissemination system, in: *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking* (1999) pp. 163–173.

[8] Q. Hu, W.-C. Lee and D.L. Lee, Indexing techniques for wireless data broadcast under data clustering and scheduling, in: *Proceedings of the Eighth International Conference on Information and Knowledge Management* (November 1999) pp. 351–358.

[9] T. Imielinski, S. Viswanathan and B. Badrinath, Data on air: organization and access, IEEE Transactions on Knowledge and Data Engineering 9(3) (June 1997) 353–372.

[10] J. Jing, A. Helal and A. Elmagarmid, Client–server computing in mobile environments, ACM Computing Surveys 31(2) (June 1999) 117–157.

[11] R.C.T. Lee, R.C. Chang, S.S. Tseng and Y.T. Tsai, *Introduction to the Design and Analysis of Algorithms* (Unalis Press).

[12] W.-C. Lee and D.-L. Lee, Signature caching techniques for information filtering in mobile enviroments, ACM Journal of Wireless Networks 5(l) (January 1999) 57–67.

[13] S.-C. Lo and A.L.P. Chen, Optimal index and data allocation in multiple broadcast channels, in: *Proceedings of the 16th International Conference on Data Engineering* (March 2000) pp. 293–302.
[14] W.-C. Peng and M.-S. Chen, Dynamic generation of data broadcasting programs for a broadcast disk array in a mobile computing environment, in: *Proceedings of the ACM 9th International Conference on Information and Knowledge Management* (November 2000) pp. 38–45.
[15] E. Pitoura and P.K. Chrysanthis, Exploiting versions for handling updates in broadcast disks, in: *Proceedings of 25th International Conference on Very Large Data Bases* (September 1999) pp. 114–125.
[16] K. Prabhakara, K.A. Hua, and J.-H. Oh, Multi-level multi-channel air cache designs for broadcasting in a mobile environment, in: *Proceedings of the 16th International Conference on Data Engineering* (February 2000) pp. 167–176.
[17] N. Shivakumar and S. Venkatasubramanian, Energy efficient indexing for information dissemination in wireless systems, ACM Journal of Wireless Networks and Applications 1(4) (January 1996) 433–446.
[18] K. Stathatos, N. Roussopoulos and J.S. Baras, Adaptive data broadcast in hybrid networks, in: *Proceedings of the 23rd International Conference on Vary Large Data Bases* (August 1997) pp. 326–335.
[19] C.-J. Su and L. Tassiulas, Broadcast scheduling for information distribution, in: *Proceedings of the 6th IEEE International Conference on Information and Communication* (April 1997) pp. 109–117.
[20] C.-J. Su and L. Tassiulas, Joint broadcast scheduling and user's cache management for efficient information delivery, in: *Proceedings of the 4th ACM/IEEE International Conference on Mobile Computing and Networking* (October 1998) pp. 33–42.
[21] WAP application in Nokia, `http://www.nokia.com/corporate/wap/future.html`
[22] WAP application in Unwired Planet, Inc., `http://phone.com`
[23] J.X. Yu, T. Sakata and K. Tan, Statistical estimation of access frequencies in data broadcasting environments, ACM/Baltzer Wireless Networks 6(2) (March 2000) 89–98.

Wen-Chih Peng biography and photo not available at time of publication.
E-mail: wcpeng@arbor.ee.ntu.edu.tw

Jiun-Long Huang biography and photo not available at time of publication.
E-mail: jlhuang@arbor.ee.ntu.edu.tw

Ming-Syan Chen biography and photo not available at time of publication.
E-mail: mschen@cc.ee.ntu.edu.tw