Wireless Networks 9, 117–129, 2003

Efficient Channel Allocation Tree Generation for Data Broadcasting in a Mobile Computing Environment

WEN-CHIH PENG 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 continuously and repeatedly broadcasts data to mobile users. These broadcast channels are also known as "broadcast disks" from which mobile users can retrieve data. Using broadcasting, mobile users can obtain the data of interest efficiently and only need to wait for the required data to present on the broadcast channel. The issue of designing proper data allocation in the broadcast disks is to reduce the average expected delay of all data items. We explore in this paper 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, we first transform the problem of generating hierarchical broadcast programs into the one of constructing a channel allocation tree with variant-fanout. By exploiting the feature of tree generation with variant-fanout, we develop a heuristic algorithm VF^K to minimize the expected delay of data items in the broadcast program. In order to evaluate the solution quality obtained by algorithm VF^K and compare its resulting broadcast program with the optimal one, we devise an algorithm OPT based on a guided search to obtain the optimal solution. Performance of these algorithms is comparatively analyzed. Sensitivity analysis on several parameters, including the number of data items and the number of broadcast disks, is conducted. It is shown by our simulation results that by exploiting the feature of variant-fanout in constructing the channel allocation tree, the solution obtained by algorithm VF^K is of very high quality and is in fact very close to the optimal one resulted by algorithm OPT. Moreover, algorithm VF^K is of very good scalability which is important for algorithm VF^K to be of practical use to generate hierarchical broadcast programs dynamically in a mobile computing environment.

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 [26,27]. 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 [5,9,15,28].

In order to conserve the energy and communication bandwidth of a mobile computing system, a data delivery architecture in which a server continuously and repeatedly broadcasts data to a client community through a broadcast channel was proposed in [1,13,14,17,25]. In a push-based information system, a server generates a broadcast program to broadcast data to the mobile users. This broadcast channel is referred to as a "broadcast disk" from which mobile clients can retrieve data. The mobile users need to wait for the data of interest to appear on the broadcast channel, and 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 previously addressed by other researchers [3,20,22,24]. Also, a significant amount of research effort has been elaborated on developing the index mechanism in multiple broadcast channels [18,21]. 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. Organizing data in a broadcast disk array raises a number of new research problems. The most important issue is 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. 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 figure 1 where two broadcast programs are presented. Assume that the data items R_i, $1 \leqslant i \leqslant 6$, are of the same size and figure 1(a) shows a flat broadcast program, where data items are evenly allocated to the two broadcast disks and the speeds of two broadcast disks in the broadcast

* Corresponding author.

Table 1
Expected delays and access frequencies of data items under two broadcast programs.

(a)

	Expected delay of data items					
	d_{R_1}	d_{R_2}	d_{R_3}	d_{R_4}	d_{R_5}	d_{R_6}
In figure 1(a)	1	1	1	1	1	1
In figure 1(b)	0.5	0.5	1.5	1.5	1.5	1.5

(b)

	Access frequency						Average expected delay	
	$P_r(R_1)$	$P_r(R_2)$	$P_r(R_3)$	$P_r(R_4)$	$P_r(R_5)$	$P_r(R_6)$	in figure 1(a)	in figure 1(b)
Case 1	0.167	0.167	0.167	0.167	0.167	0.167	1	1.16
Case 2	0.25	0.25	0.125	0.125	0.125	0.125	1	1
Case 3	0.3	0.3	0.1	0.1	0.1	0.1	1	0.9
Case 4	0.4	0.4	0.05	0.05	0.05	0.05	1	0.7

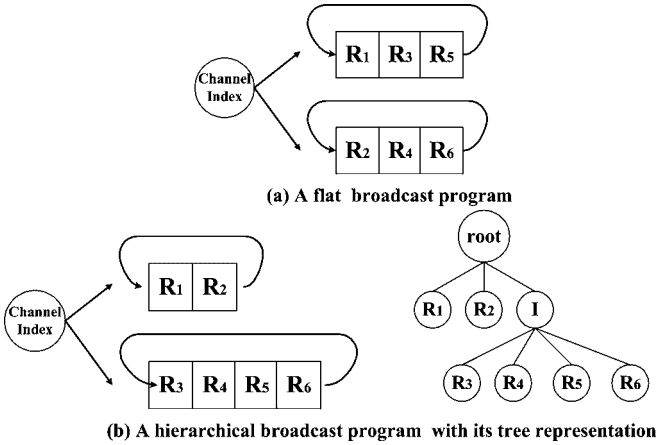

Figure 1. Two broadcast programs for the broadcast disk array of two broadcast disks.

disk array are the same, meaning that the expected delays for all data items are equal from one to another. In contrast, figure 1(b) shows another broadcast program with its channel allocation tree (or abbreviatedly as allocation tree). As will be described in section 2 later, 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. In figure 1(b), the upper channel is allocated with two data items and the lower channel is allocated with four data items. Notice that in figure 1(b) the data items in the fast disk (i.e., the upper broadcast channel) spin twice as fast as those data items in the slow disk (i.e., the lower broadcast channel). The allocation tree in figure 1(b) is called an allocation tree with variant-fanout in this paper. Denote the expected delay and the access frequency of data item R_i, respectively, as d_{R_i} and $P_r(R_i)$. Table 1(a) shows the expected delay for data items under the two different allocations of broadcast arrays, and four sets of access frequencies of data items are given in table 1(b) for illustrative purposes. The average expected delay in table 1(b) 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., $\sum_{i=1}^{6} d_{R_i} \cdot P_r(R_i)$. Same as in [1,23], 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 i is the number of data items allocated in the broadcast disk i. For example, for the allocation in figure 1(a), $N_1 = 3$ and $N_2 = 3$, and for the allocation in figure 1(b), $N_1 = 2$ and $N_2 = 4$. For ease of exposition, the details of formulating the average expected delay, which are in essence the same as those used in [1,23], are given in the appendix for interested readers. It can be verified that the expected delays of data items R_1 and R_2 in figure 1(a) are both equal to $d_{R_1} = d_{R_2} = (2 + 1 + 0)/3 = 1$. On the other hand, the expected delays of data items R_1 and R_3 in figure 1(b) are $d_{R_1} = (1 + 0)/2 = 0.5$ and $d_{R_3} = (3 + 2 + 1 + 0)/4 = 1.5$, respectively.[1]

It can be seen from table 1(b) that when the access frequencies of all data items are the same, the flat broadcast program in figure 1(a) is the one to use, and we do not want to allocate different numbers of data items to different broadcast disks as in figure 1(b). Explicitly, the average expected delay of data items for case 1 is $\sum_{i=1}^{6} d_{R_i} \cdot P_r(R_i) = 1 \cdot 0.167 + 1 \cdot 0.167 + 1 \cdot 0.167 + 1 \cdot 0.167 + 1 \cdot 0.167 + 1 \cdot 0.167 = 1$, whereas that in figure 1(b) is $\sum_{i=1}^{6} d_{R_i} \cdot P_r(R_i) = 0.5 \cdot 0.167 + 0.5 \cdot 0.167 + 1.5 \cdot 0.167 + 1.5 \cdot 0.167 + 1.5 \cdot 0.167 + 1.5 \cdot 0.167 = 1.16$. It can be verified that the average expected delay resulting from the flat broadcast program will remain the same as the access frequencies of data items vary. However, when the access frequencies become increasingly skewed, the average expected delay of allocation with variant-fanout is reduced from 1.16 in case 1 to 0.7 in case 4. With the access frequencies being skewed, the average expected delay of the hierarchical broadcast program based on variant-fanout becomes much smaller than that of the flat broadcast program, showing the very advantage of exploiting the feature of hierarchy in designing broadcast programs for a broadcast disk array.

Consequently, we explore in this paper the issue of generating hierarchical broadcast programs with the data access

[1] Note that as described in the appendix, same as in prior work [1,23], the expected delay d_{R_i} we consider in this paper does not include the residual waiting time which is half of the size of a data item. This provision will facilitate our presentation, but will not affect the quality of solutions derived.

frequencies and the number of broadcast disks in a broadcast disk array given. Specifically, we first formulate the problem of generating hierarchical broadcast programs and transform this problem into the one of constructing a channel allocation tree with variant-fanout. A hierarchical broadcast program for the broadcast disk array of K broadcast disks can then be represented by a channel allocation tree with a height of K. By exploiting the feature of tree generation with variant-fanout, we develop a heuristic algorithm VF^K (standing for variant-fanout with the constraint K), which is basically a family of algorithms with different values of K, to minimize the expected delay of the corresponding broadcast program. In order to evaluate the solution quality obtained by algorithm VF^K and compare its resulting broadcast program with the optimal one, we devise an algorithm OPT (standing for OPTimal) to obtain the optimal solution. Algorithm OPT is mainly a guided search and is similar to the well-known A* search by using a cost function to guide the search and to ensure the optimality of the goal node reached [19]. With the proper design of the guide function, algorithm OPT can obtain the optimal solution very efficiently. Performance of these algorithms is comparatively analyzed and sensitivity analysis on several parameters, including the number of data items and the number of broadcast disks, is conducted. It is shown by our simulation results that by exploiting the feature of variant-fanout in constructing the channel allocation tree, the solution obtained by algorithm VF^K is of very high quality and is in fact very close to the optimal one resulted by algorithm OPT. With its polynomial time complexity, algorithm VF^K incurs a much shorter execution time than algorithm OPT. As the number of broadcast disks and the number of data items increase, the advantage of algorithm VF^K over algorithm OPT becomes more prominent. It is worth mentioning that even when the number of broadcast channels K changes dynamically, algorithm VF^K can reach the new configuration very efficiently with the number of data items required to be moved around broadcast channels minimized, showing another advantage of algorithm VF^K. It is also shown by our experimental results that algorithm VF^K is not only able to produce the solutions of very high quality but also of good scalability which is important for VF^K to be of practical use to generate hierarchical broadcast programs dynamically in a mobile computing environment.

A significant amount of research effort has been elaborated upon issues of data broadcast [1,3,4,12,24]. We mention in passing that the authors in [1] explored a push-based data delivery architecture using the broadcast disk to meet the need of mobile applications. The research on striking a balance between push and pull data delivery methods was conducted in [2]. The authors in [11] proposed a mechanism to capture data access patterns which can be utilized in a data delivery model composed of push-based and pull-based delivery methods. Without fully exploiting the advantage of multiple broadcast channels, the attention of those studies in [1,6,8,11,18] was mainly paid to the design of data delivery model and index methods for a single broadcast channel, but not for multiple broadcast channels. The design of index methods in multiple broadcast channels was addressed in [18,21], where, however, the problem of generating hierarchical broadcast programs was not considered.

The contributions of this paper are twofold. First, we propose an innovative concept to broadcast data in a hierarchical manner through multiple broadcast channels, and then, in light of this concept, we develop algorithm VF^K which utilizes the feature of variant-fanout tree generation to minimize the average expected delay of data items. In addition, in order to evaluate the quality of the solution obtained by algorithm VF^K, we develop algorithm OPT which exploits the feature of A* search to obtain the optimal solution for performance comparison with algorithm VF^K. To the best of our knowledge, priori works neither fully explored the impact of using a broadcast disk array, nor utilized the feature of hierarchy to allocate data items for broadcasting, let alone developing efficient algorithms to generate hierarchical broadcast programs and conducting their performance studies. These features distinguish this paper from others. The fast increase in mobile applications which use broadcasting for information dissemination justifies the timeliness and importance of this study.

This rest of this paper is organized as follows. Preliminaries are given in section 2. In section 3, we develop algorithm VF^K for allocating data items to a broadcast disk array. We devise algorithm OPT in section 4 to obtain the optimal solution. Performance studies are conducted in section 5. This paper concludes with section 6.

2. Preliminaries

Note that as the number of data items in a broadcast disk increases, the expected delay of those data items within the broadcast disk increases. Theoretically, generating such 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 of minimizing the average expected delay of all data items. Table 2 shows the description of symbols used in modelling the program. Figure 2 shows the program formulation of generating a hierarchical broadcast program for a broadcast disk array of K broadcast disks. Denote the total number of

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	$P_r(R_j)$
The expected delay of data item R_j	d_{R_j}
The number of leaf nodes in level i of the allocation tree	L_i
The weight of leaf nodes in level i of the allocation tree	w_i
The aggregate access frequency for the leaf nodes in level i	P_{L_i}

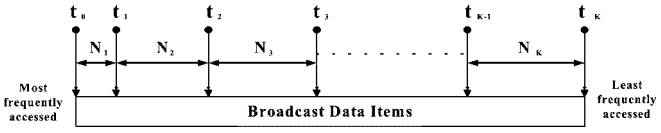

Figure 2. Generating hierarchical broadcast programs for a broadcast disk array of K broadcast disks, where N_i is the number of data items allocated to disk i and $t_i = \sum_{j=1}^{i} N_j$.

data items as n, and a data item as R_i, $1 \leqslant i \leqslant n$. Recall that $P_r(R_i)$ is the access frequency of R_i and $\sum_{i=1}^{n} P_r(R_i) = 1$. Assume that all data items in figure 2 have been sorted according to the descending order of the access frequencies of all data items. As can be seen in figure 2, a broadcast disk array of K broadcast disks corresponds to the partition of n data items into K groups, where N_i is the number of data items to be allocated to broadcast disk i. We then have $\sum_{i=1}^{K} N_i = n$. According to the characteristics of broadcast disks, we have the following properties.

Property 1. The expected delay of the data items within the broadcast disk i, denoted by d_i, is $d_i = \sum_{j=1}^{N_i}(N_i - j)/N_i$, where N_i is the number of data in that broadcast disk.

Property 2. Let $t_i = \sum_{h=1}^{i} N_h$ and $t_0 = 0$. Then, N_i data items, denoted by R_j, $t_{i-1} + 1 \leqslant j \leqslant t_i$, are allocated to broadcast disk i, and $d_i = d_{R_j}$ for $j \in [t_{i-1} + 1, t_i]$.

For example, in figure 1(b), $d_1 = d_{R_1} = d_{R_2} = 0.5$ since R_1 and R_2 are in broadcast disk 1, and $d_2 = d_{R_3} = d_{R_4} = d_{R_5} = d_{R_6}$ since R_3, R_4, R_5 and R_6 are in broadcast disk 2. The problem that we study in this paper can be formally defined as follows:

Problem of generating a hierarchical broadcast program. 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 of minimizing the average expected delay of all data items. The average expected delay of all data items in a broadcast disk array of K broadcast disks can be formulated as follows:

$$\begin{aligned}
&\sum_{j=1}^{n} d_{R_j} \cdot P_r(R_j) \\
&\quad = \sum_{i=1}^{K} d_i \sum_{j=t_{i-1}+1}^{t_i} P_r(R_j) \\
&\quad = \sum_{i=1}^{K} \left(\sum_{q=1}^{N_i} \frac{N_i - q}{N_i} \right) \sum_{j=t_{i-1}+1}^{t_i} P_r(R_j), \qquad (1)
\end{aligned}$$

where $t_i = \sum_{h=1}^{i} N_h$ and $t_0 = 0$.

Problem transformation. As mentioned above, generating such a broadcast program can be viewed as a partition problem. We now would like to transform this partition problem

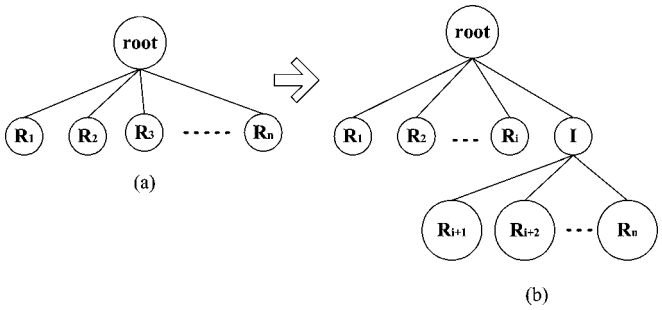

Figure 3. Grouping a set of nodes and moveing them to a lower level to reduce the weights of leaf nodes.

on data items into the one of generating a channel allocation tree. The initial allocation tree is mainly a tree of all data items attached to the root node (as shown in figure 3(a)). 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. Thus, the allocation tree in figure 3(a) corresponds to the case that all data items are put in one broadcast disk. We next expand the allocation tree by moving a set of nodes to one level lower as shown in figure 3(b) (the criterion to determine such a set of nodes will be explained in section 3 later). Figure 3(b) corresponds to the case that two broadcast disks are used to store the data items. This procedure continues by performing more partitions on data items until all broadcast disks in the broadcast disk array are used. As such, a hierarchical broadcast program for a broadcast disk array of K broadcast disks can be represented as a channel allocation tree with a height of K.

It can be seen that similar to the formulation of the expected delay for those data items residing in a broadcast disk, the weight of leaf nodes in level i, denoted by w_i, can be formulated as $w_i = \sum_{x=1}^{L_i}(L_i - x)/L_i$, where L_i is the number of leaf nodes in the level i. The weight w_i in fact corresponds to the expected delay of each data item in level i. Explicitly, when the construction of an allocation tree is completed, L_i equals N_i and w_i equals d_i. Let P_{L_i} be the summation of access frequencies of data items associated with leaf nodes in level i. For example, it can be seen from figure 1(b) that the depth of the allocation tree is 2 and those leaf nodes in the same level are allocated in the same broadcast disk. For that allocation tree with access frequencies given in case 3 of table 1(b), we have $w_1 = 0.5$, $w_2 = 1.5$, $P_{L_1} = 0.6$, and $P_{L_2} = 0.4$. According to the formulation of average expected delay for broadcast disk array, we have the cost of the tree below:

$$\sum_{i=1}^{K} w_i P_{L_i} = \sum_{i=1}^{K} \left(\sum_{q=1}^{L_i} \frac{L_i - q}{L_i} \right) \cdot P_{L_i}, \qquad (2)$$

where

$$P_{L_i} = \sum_{j=s_{i-1}+1}^{s_i} P_r(R_j),$$

$$s_i = \sum_{h=1}^{i} L_h \quad \text{and} \quad s_0 = 0.$$

As such, the problem of generating a hierarchical broadcast program can be transformed into the one of building an allocation tree with the minimal cost.

3. Algorithm VF^K: using variant-fanout for allocation tree generation

We devise in section 3.1 a heuristic algorithm VF^K to generate the channel allocation tree which explores the feature of tree generation with variant-fanout to minimize the expected delay of the corresponding broadcast program. The execution scenario of algorithm VF^K is illustrated in section 3.2.

3.1. Design of algorithm VF^K

With the problem transformation described above, we devise algorithm VF^K (standing for Variant-Fanout with the constraint K) to explore variant-fanout in the allocation tree generation to minimize the cost of this tree.

Algorithm VF^K is greedy in nature and builds the allocation tree in a top down manner. Algorithm VF^K starts with attaching all data records to the root node. Then, after some evaluation, algorithm VF^K groups nodes with small access frequencies and moves them to one level lower so as to reduce the cost of the tree. Figure 3 shows the scenario of grouping a set of nodes and moving them to a lower level.

Definition 1. Suppose that level v in the allocation tree has $j - i + 1$ data nodes, $R_i, R_{i+1}, \ldots, R_j$. The cost of level v is defined as

$$C_{i,j} = \sum_{k=1}^{j-i+i} \frac{(j - i + 1) - k}{j - i + 1} \sum_{q=i}^{j} P_r(R_q),$$

which is equal to $w_v \cdot P_{L_v}$, where w_v and P_{L_v} are, respectively, the weight of leaf nodes and the aggregate access frequency for the leaf nodes in level v of the allocation tree.

In essence, the value of $C_{i,j}$ is related to the average expected delay of leaf nodes in level v. It can be seen from figure 3 that moving down those nodes to the next level decreases the corresponding expected delay of leaf nodes, and thus the cost of the allocation tree will be reduced. In order to evaluate the reduction of partitioning, we have the following definition.

Definition 2. Suppose that node R has $j - i + 1$ child data nodes, $R_i, R_{i+1}, \ldots, R_j$, which are sorted according to the descending order of $P_r(R_q)$, $i \leqslant q \leqslant j$, i.e., $P_r(R_q) \geqslant P_r(R_y)$ iff $q \leqslant y$. The reduction gain achieved by grouping nodes $R_{p+1}, R_{p+2}, \ldots, R_j$ and attaching them under a new child node, denoted by $\delta(p)$, can be formulated as $\delta(p) = C_{i,j} - (C_{i,p} + C_{p+1,j})$.

Algorithm VF^K.

Input. Assume that $R_1, \ldots, R_n$ have been sorted according to the descending order of $P_r(R_j)$, $1 \leqslant j \leqslant n$, i.e., $P_r(R_q) \geqslant P_r(R_y)$ iff $q \leqslant y$. K is the number of broadcast disks in a broadcast disk array.
Output. The resulting allocation tree.
begin
1. Create table AT with K rows;
2. $AT(1).B = 1$; /* $AT(1).B$ records the beginning of level 1 */
3. $AT(1).E = n$; /* $AT(1).E$ records the end of level 1 */
4. $AT(1).LC = C_{1,n}$; /* $AT(1).LC$ records the cost of level 1 */
5. **for** each row i in table AT and $i \geqslant 2$
6. **begin**
7. $AT(i).B = 0$; /* $AT(i).B$ records the beginning of level i */
8. $AT(i).E = 0$; /* $AT(i).E$ records the end of level i */
9. $AT(i).LC = 0$; /* $AT(i).LC$ records the cost of level i */
10. **end**
11. $pivot = 1$;
12. **repeat**
13. **begin**
14. Choose row i from table AT such that $AT(i).LC$ is maximal among all unmarked rows;
15. **if** ($i == 1$ or $i == pivot$) /* Upward and downward partition */
16. **begin**
17. $j = Partition(R_{AT(i).B}, R_{AT(i).B+1}, \ldots, R_{AT(i).E})$;
18. {Update table AT accordingly and unmark all rows;
19. $pivot++$;}
20. **end**
21. **else** /* Middle partition */
22. **begin**
23. $j = Partition(R_{AT(i).B}, R_{AT(i).B+1}, \ldots, R_{AT(i).E})$;
24. **if** $(AT(i-1).E - AT(i-1).B) < (j - AT(i).B)$
25. {Update table AT accordingly and unmark all rows;
26. $pivot++$;}
27. **else** {
28. Mark row i;
29. Merge $(R_{AT(i).B}, R_{AT(i).B+1}, \ldots, R_j)$ and $(R_{j+1}, R_{j+2}, \ldots, R_{AT(i).E})$ together;}
30. **end**
31. **end**
32. **until** ($pivot == K$)
end

In light of definition 2, we devise algorithm VF^K which contains a procedure *Partition* to identify the group of nodes to be moved downward in each execution level so as to maximize the reduction gain in each step.

Procedure *Partition*$(R_i, R_{i+1}, \ldots, R_j)$.

1. Determine p^* such that
$$\delta(p^*) = \max_{\forall p \in \{i, i+(j-i+1)/2-1\}} \{\delta(p)\}.$$
2. Attach nodes $R_{p^*+1}, R_{p^*+2}, \ldots, R_j$ under a new node I in the tree.
3. Return p^*.

To generate the allocation tree, algorithm VF^K first starts with a configuration where all nodes are attached to the root. Table AT (standing for Auxiliary Table) is created to record the status of the allocation tree as stated from line 1 to line 10 in algorithm VF^K. Note that since there are three kinds of

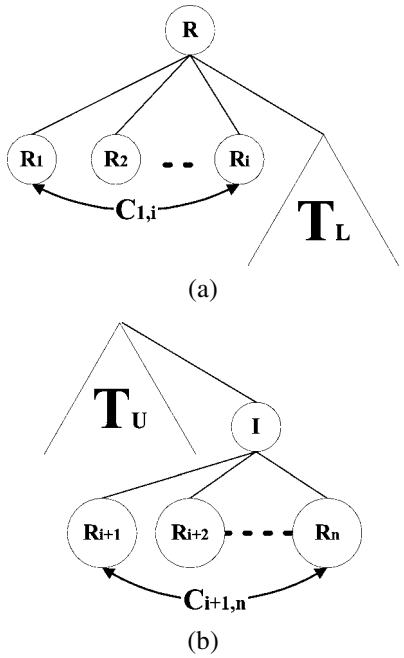

Figure 4. The tree representations by using T_{L} and T_{U}. (a) A tree representation by using T_{L} subtree. (b) A tree representation by using T_{U} subtree.

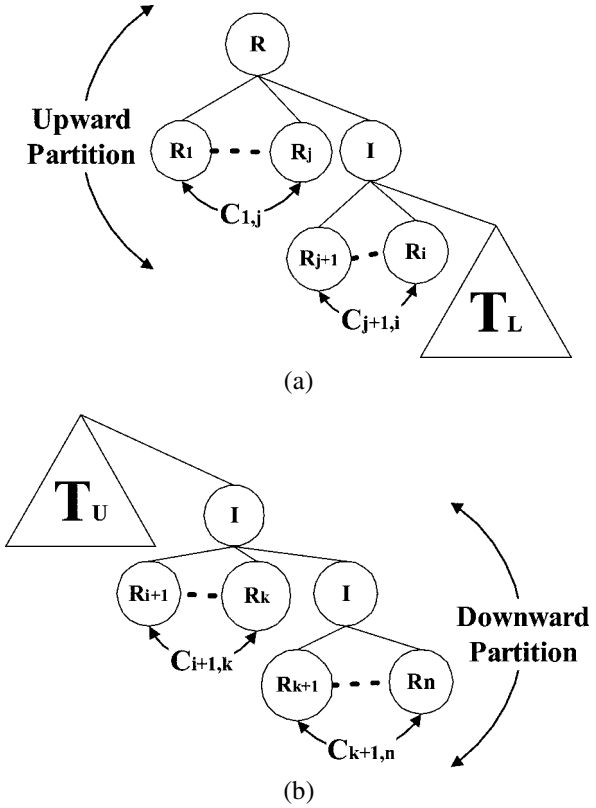

Figure 5. Illustrations of upward and downward partitions. (a) A scenario of upward partition. (b) A scenario of downward partition.

partitioning, i.e., upward, middle and downward partitions, judiciously applying these partitions is able to reduce the total cost of the allocation tree. To facilitate the description of algorithm VF^K, we define two subtrees, i.e., T_U and T_L, where T_U (respectively, T_L) is left-upper (respectively, right-lower) subtree and does not contain the top level (respectively, the last level) of the tree. Consider the tree representations by T_U and T_L in figure 4 where the original tree is the one in figure 3(b). Figure 5(a) shows the scenario of an upward partition for the allocation tree in figure 4(a). On the other hand, figure 5(b) illustrates the scenario of a downward partition for the allocation tree in figure 4(b). From table AT, algorithm VF^K chooses the level with the maximal cost to partition (line 14 of algorithm VF^K). Let *pivot* be the number of levels expanded in the allocation tree thus far. Explicitly, as can be seen from line 15 to line 20 in algorithm VF^K, if the cost of the top (respectively, the last) level is larger than that of other levels, algorithm VF^K performs upward (respectively, downward) partition. Otherwise, VF^K performs the middle partition according to the operations from line 21 to line 31 in algorithm VF^K. Table AT is then updated accordingly. As can be seen from line 24 to line 29 in algorithm VF^K, algorithm VF^K makes sure that each partition will satisfy the feature of hierarchy. That is, the number of leaf nodes (i.e., data items) in the upper level is always smaller than that in the lower level. According to definition 2, the candidate set of nodes with the maximal reduction gain $\delta(p^*)$ is chosen. When such a set of nodes is identified and moved to the next level, those nodes will be evaluated by themselves to see if any further partition for some of them is necessary in line 32 (the condition ($pivot == K$) means that the number of the channels has been reached). Algorithm VF^K partitions the nodes iteratively with the objective of minimizing the average cost $\sum_{i=1}^{K} w_i \cdot P_{L_i}$ until the depth of the tree reaches the number of broadcast disks in the broadcast disk array. As such, the allocation tree is expanded and constructed in a top down manner.

It can be verified that algorithm VF^K is of polynomial time complexity. With n sorted data items and a broadcast disk array of K broadcast disks given, the complexity of algorithm VF^K can be expressed by $K \cdot (O(K \log K) + O(n))$. Specifically, the complexity of choosing the partition with the maximal cost from table AT is $K \cdot O(K \log K)$ and that of partitioning is $K \cdot O(n)$. When the number of data items is significantly larger than the number of broadcast disks, i.e., $n \gg K$, the value of $K \cdot O(n)$ is the dominating factor of the complexity of VF^K.

3.2. An example execution scenario VF^K

For example, consider the profile in table 3 where the number of data items n is 11 and the number of broadcast disks K is 4. The initial tree configuration is shown in figure 6(a), where all data records are attached to the root. Procedure *Partition* then determines the optimal partition of nodes to be moved to the next level. The values in table AT and their changes made in accordance with the execution of algorithm VF^K are shown in table 4, where in table 4(a)–4(c), the first table is table AT and the second table is the one to determine the cut point p^* for the partition selected where the maximal value of $\delta(p)$ is marked with an '*'. From the calculation shown in table 4(a), we obtain $p^* = 4$, and therefore group nodes $R_5, R_6, \ldots,$ and

Table 3
The profile of an illustrative example.

Data record	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}
$P_r(R_i)$	0.237	0.211	0.132	0.132	0.08	0.05	0.05	0.027	0.027	0.027	0.027

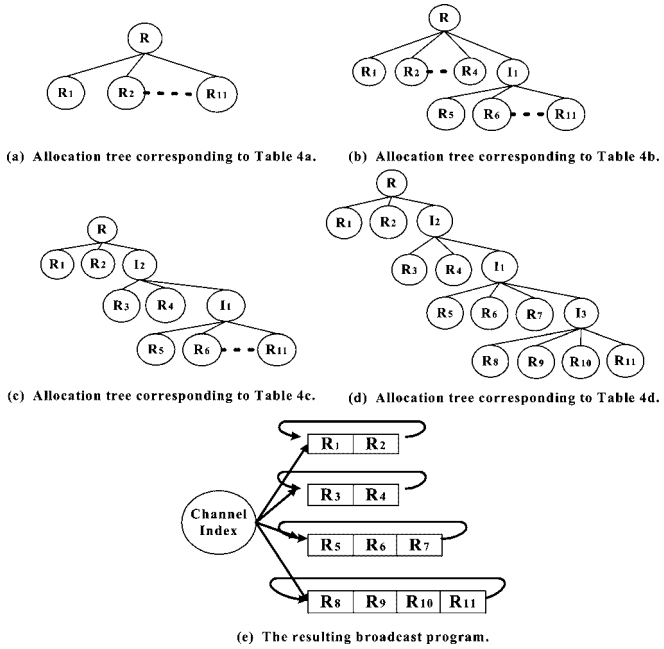

Figure 6. An execution scenario of algorithm VF^K: (a)–(d) the generation of the allocation tree, and (e) the resulting broadcast program.

R_{11} together and move them to the next level, resulting in the configuration shown in figure 6(b). Note that in table AT, the value of $AT(i)$.LC is the cost of nodes in level i, i.e., $w_i \cdot P_{L_i}$, and explicitly, equal to $C_{AT(i).B,\ AT(i).E}$. In table 4(b), since $AT(1).LC > AT(2).LC$, the upward partition is performed. Following the calculation shown at the bottom of table 4(b), we have $p^* = 2$ and figure 6(b), in turn, leads to figure 6(c). Similarly, we obtain $AT(1).LC = 0.224$ and $AT(2).LC = 0.132$ shown in table 4(c). Since $AT(3).LC = 0.864$ is the maximal in table 4(c), algorithm VF^K performs the downward partition. Following the calculation in table 4(c), we have $p^* = 7$. Then, the configuration shown in figure 6(d) and in table 4(d) follows. As the depth of the allocation tree equals the number of broadcast disks, algorithm VF^K completes. It can be seen that the very advantage of algorithm VF^K is that even when the number of broadcast channels K changes dynamically, algorithm VF^K can reach the new configuration very efficiently with the number of data items required to be moved around broadcast channels minimized. Also, as validated in our experiments in section 5, when the number of data items to be broadcast is large, algorithm VF^K can expand the allocation tree very efficiently. According to the configuration in figure 6(d), we can have the hierarchical broadcast program shown in figure 6(e) where it can be verified that the average expected delay is the sum of costs of all levels in the tree shown in figure 6(d), i.e., $0.224 + 0.132 + 0.18 + 0.162 = 0.698$.

Table 4
Determining from table AT the set of nodes to be grouped together as a partition for allocation tree generation.

(a) Partition $(R_1, R_2, \ldots, R_{11})$ is selected and decomposed to $(R_1, \ldots, R_4)$ and $(R_5, \ldots, R_{11})$.

Level i	$AT(i).B$	$AT(i).E$	$AT(i).LC$
1	1	11	5*
2	0	0	0
3	0	0	0
4	0	0	0

p	1	2	3	4	5
$C_{1,11}$	5	5	5	5	5
$C_{1,p} + C_{p+1,11}$	3.4335	2.484	2.05	1.932	2.104
$\delta(p)$	1.5665	2.516	2.95	3.068*	2.896

(b) Partition $(R_1, R_2, \ldots, R_4)$ is selected and decomposed to (R_1, R_2) and (R_3, R_4).

Level i	$AT(i).B$	$AT(i).E$	$AT(i).LC$
1	1	4	1.068*
2	5	11	0.864
3	0	0	0
4	0	0	0

p	1	2
$C_{1,4}$	1.068	1.068
$C_{1,p} + C_{p+1,4}$	0.475	0.356
$\delta(p)$	0.593	0.712*

(c) Partition $(R_5, R_6, \ldots, R_{11})$ is selected and decomposed to $(R_5, \ldots, R_7)$ and $(R_8, \ldots, R_{11})$.

Level	$AT(i).B$	$AT(i).E$	$AT(i).LC$
1	1	2	0.224
2	3	4	0.132
3	5	11	0.864*
4	0	0	0

p	5	6	7
$C_{5,11}$	0.864	0.864	0.864
$C_{5,p} + C_{p+1,11}$	0.52	0.381	0.342
$\delta(p)$	0.344	0.483	0.522*

(d) The final result of table AT.

Level i	$AT(i).B$	$AT(i).E$	$AT(i).LC$
1	1	2	0.224
2	3	4	0.132
3	5	7	0.18
4	8	11	0.162

4. Algorithm OPT: obtaining the optimal broadcast program

In order to compare the solution quality obtained by algorithm VF^K with the optimal one, by utilizing the concept of A* search [19], we implement in section 4.1 algorithm OPT which constructs a solution tree and is able to determine the optimal solution. The execution scenario of algorithm OPT is presented in section 4.2.

4.1. Design of algorithm OPT

Finding solutions for the problem of partitioning data items and allocating them into each broadcast disk can be represented by a search problem based on state transition. In essence, algorithm OPT is a best-first state transition search [16,19]. The search made by algorithm OPT can be represented by a solution tree where each node is associated with a state of partitioning. Note that the solution tree in algorithm OPT should not be confused with the allocation tree in algorithm VF^K. Figure 7 shows part of an example state transition search which corresponds to the case of allocating 6 data items into 3 broadcast disks. Starting from the root in the level one, algorithm OPT generates nodes in level i to explore the possible partitions of data items into i group. Algorithm OPT is mainly a guided search and is similar to the well-known A* search by using a cost function to guide the search and to ensure the optimality of the goal node reached. However, algorithm OPT we employed in this study is different from a typical A* search in that our interest here is only limited to obtain the optimal partition efficiently so that we can evaluate the solution quality obtained by algorithm VF^K. Yet, in this specific application, we do not have to use the optimal path from the root to the goal node found by the search.

As shown in figure 7, a node in the solution tree contains the set of partitions where the data items in the same partition are allocated to the same broadcast disk. A node in the level j has j partitions, which is associated with one allocation for a broadcast disk array of j broadcast disks. The set of partitions for the node i in the level j is denoted by SP_i and the partitions belonging to SP_i are denoted by $P_1, P_2, \ldots, P_j$. We use $|SP_i|$ to represent the number of partitions in that set. Let the data items in the beginning and the end of P_j be denoted by $P_j.B$ and $P_j.E$, respectively. The cost of the partition P_j, expressed by $c(P_j)$, is then $C_{P_j.B,P_j.E}$ which can be determined by definition 1. The cost of the node can be obtained by summing up the costs of partitions within that node. Also, let $P_{\max}$ refer to the partition having the maximal cost within the node. Consider the node marked with a star in figure 7 as an example. In that node, there are three partitions P_1, P_2 and P_3. It can be verified that $c(P_1) = C_{1,1} = 0$, $c(P_2) = C_{2,2} = 0$, $c(P_3) = C_{3,6}$ and the cost of that node is equal to the value of $\sum_{i=1}^{3} c(P_i)$. P_3 is then the $P_{\max}$ of that node with $P_{\max}.B = 3$ and $P_{\max}.E = 6$. With a given solution tree of its depth K, the problem of generating a hierarchical broadcast program for K broadcast disks becomes a search problem based on state transition in which a node in level K with the minimal cost is to be found (i.e., the goal node).

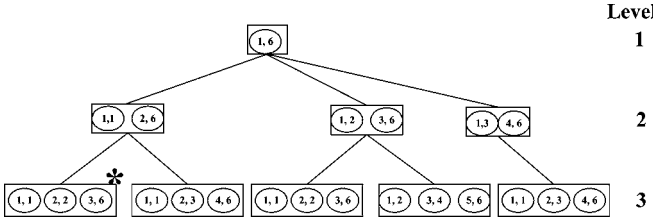

Figure 7. The solution tree with the number of data items to be 6 for $K = 3$.

Similarly to an A* search, the search in algorithm OPT is controlled by an evaluation function $f(\cdot)$. The node i chosen for further partition (i.e., whose immediate successors will be generated in the solution tree) is the one which has the smallest value of $f(\cdot)$ among all generated nodes which have not been partitioned so far. Algorithm OPT stops the expansion until the goal node is found. Hence, the function $f(i)$ that guides the search in algorithm OPT consists of two components: the cost of reaching node i from the root, i.e., $g(i)$, and the expected cost of arriving at the goal node from node i, i.e., $h(i)$. Accordingly, $f(i) = g(i) + h(i)$.

The function $g(i)$ is calculated by the cost from the root to node i along the minimal cost path found so far in the solution tree. Assume that the number of broadcast disks in a broadcast disk array is K and the node i is in level j. Thus, we have,

$$g(i) = \sum_{x=1}^{j-1} c(P_x),$$

where $P_x \in SP_i - P_{\max}$ for the guided search in algorithm OPT.

In addition, the function $h(i)$ is the estimated cost from node i to the goal node. In order to obtain the minimal cost, $P_{\max}$ that has the maximal cost is chosen for further partitioning so as to reduce the total cost from the root to the nodes generated from node i. By utilizing the procedure *Partition* described in section 3, we have the following procedure *Partition_Cost* which returns the minimal cost for partitioning $P_{\max}$.

Procedure *Partition_Cost*($P_{\max}$).

1. $p^* = Partition(R_{P_{\max}.B}, R_{P_{\max}.B+1}, \ldots, R_{P_{\max}.E})$.
2. Return $((C_{P_{\max}.B, P_{\max}.p^*}) + (C_{P_{\max}.p^*+1, P_{\max}.E}))$.

Consequently, the function $h(i)$ for node i in level j can be defined as below:

Case 1. If the level of node i equals the number of broadcast disks, i.e., $j = K$, meaning that no further partition is needed, $h(i) = c(P_{\max})$.

Case 2. If the level of node i is smaller than the number of broadcast disks, i.e., $j < K$, meaning that further partitions are needed, $h(i) = Partition\_Cost(P_{\max})/(K - j)$.

In light of the evaluation function, algorithm OPT can be outlined below.

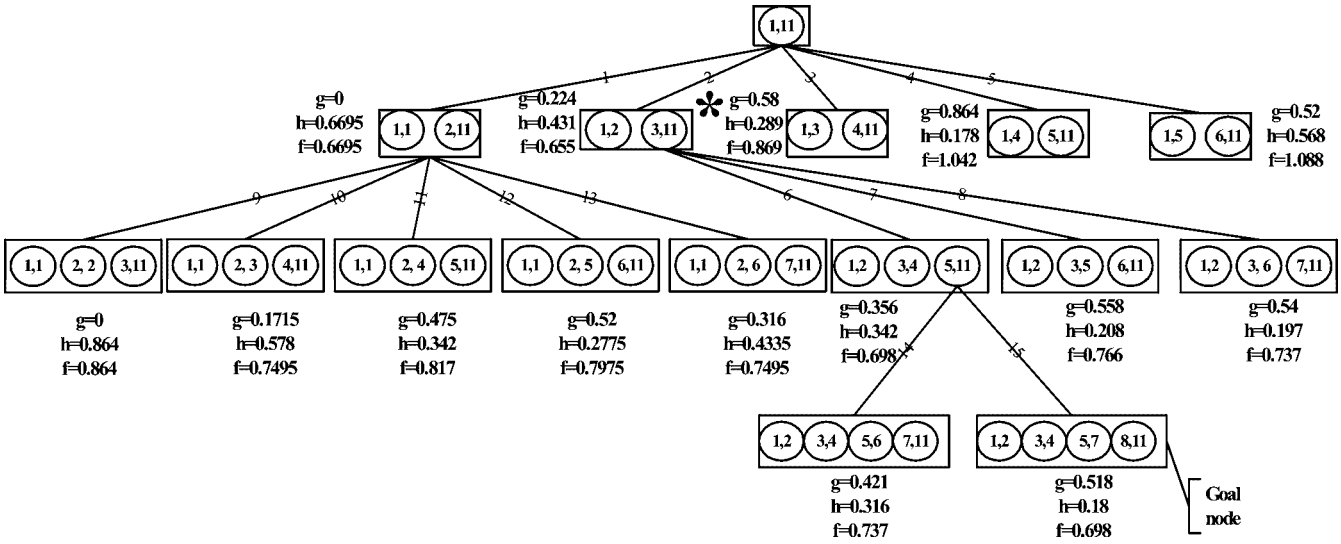

Figure 8. An execution scenario of algorithm OPT.

Algorithm OPT.

Input. Assume that $R_1, R_2, \ldots, R_n$ have been sorted according to the descending order of $P_r(R_i)$, $1 \leqslant i \leqslant n$, i.e., $P_r(R_p) \geqslant P_r(R_q)$ iff $p \leqslant q$. K is the number of broadcast disks in the broadcast disk array.
Output. Report the goal node.
begin
1. Construct a heap by the value of the evaluation function $f(\cdot)$;
   /* A heap is a data structure which returns the element with the minimal value when one is to remove an element from the heap [7]*/
2. Insert the root node into the heap;
3. **repeat**
4. **begin**
5. Remove the node i from the heap;
6. Expand the child nodes of node i;
7. Calculate the descendants of node i and insert these descendants into the heap;
8. **end**
9. **until** $(|SP_i| == K)$;
10. Return the node i;

end

Algorithm OPT first constructs the heap [7] and inserts the root node into the heap (line 1 to line 2 in algorithm OPT). Algorithm OPT removes the node i from the heap (line 5 in algorithm OPT) and expands the descendants of node i (line 6). The values of function $f(\cdot)$ for all descendants are calculated and these descendants are inserted into the heap (line 7). Algorithm OPT expands the nodes iteratively until $|SP_i| = K$ and the node i with the minimal cost is removed form the heap (line 10).

Let $h^*(i)$ denote the cost of a search path from node i to the goal node. Theorem 1 states that for any node i, $h(i) \leqslant h^*(i)$. Similarly to that in an A* search [19], this inequality ensures the optimality of the solution found by algorithm OPT.

Theorem 1. Given a node i in level j, $h(i)$ is a lower bound of $h^*(i)$.

Proof. Clearly, $h^*(i)$ can be decomposed into

$$\sum_{q=j}^{K-1} Partition\_Cost\left(P_{\max}^{q}\right) + c\left(P_{\max}^{K}\right),$$

where $P_{\max}^{q}$ is the partition with the maximal cost in level q. Note that if $(j < K)$, we have

$$\begin{aligned} &h^*(i) - h(i) \\ &\quad = (K-j)^{-1} \\ &\qquad \times \left((K-j) \left(\sum_{q=j+1}^{K-1} Partition\_Cost\left(P_{\max}^{q}\right) + c\left(P_{\max}^{K}\right) \right) \right. \\ &\qquad\quad \left. + (K-j-1) \cdot Partition\_Cost\left(P_{\max}^{j}\right) \right) \geqslant 0. \end{aligned}$$

Also, if j equals K, we have $h^*(i) - h(i) = \sum_{q=j}^{K-1} Partition\_Cost(P_{\max}^{q}) \geqslant 0$. Consequently, we have $h^*(i) \geqslant h(i)$. This theorem follows. □

4.2. An example execution scenario of OPT

Consider the profile in table 3 where the number of broadcast disks is 4, i.e., $K = 4$. The execution scenario of algorithm OPT is shown in figure 8, where the number next to each link represents the sequence of node expansion. Algorithm OPT starts with the root node and then expands the descendants. The values of functions $g(\cdot)$, $h(\cdot)$ and $f(\cdot)$ are calculated accordingly. Consider the node marked with a star in figure 8 as an example. In that node, the value of $g(\cdot)$ is $C_{1,2} = 0.224$ and that of $h(\cdot)$ equals $(C_{3,4} + C_{5,11})/2 = 0.431$. Thus,

$f(\cdot)$ is obtained accordingly, i.e., $f(\cdot) = g(\cdot) + h(\cdot) = 0.224 + 0.431 = 0.655$.

Following the same procedure, we can obtain the values of function $f(\cdot)$ for each descendant node. After expanding the root node, all these descendant nodes are inserted to the heap. Since the level of the root node is smaller than K, algorithm OPT removes from the heap the node with the minimal cost to expand. Thus, the node marked with a star in figure 8 is selected for further expansion. Algorithm OPT stops expanding after the level number reaches K. It can be seen that the resulting broadcast program, with the partition (R_1, R_2), (R_3, R_4), (R_5, R_6, R_7) and $(R_8, \ldots, R_{11})$ and the corresponding average expected delay of 0.698, obtained by algorithm OPT is indeed the same as the one obtained by algorithm VF^K in section 3.2. As the number of data items and the value of K increase, the number of nodes in the solution tree increases drastically. It can be seen that the number of nodes expanded in this illustrative example is 15. With the proper design of the guide function, OPT can obtain the optimal solution very efficiently.

Note that procedure *Partition* is the most time consuming procedure. Performance of algorithm VF^K and algorithm OPT can be compared in terms of the number of procedure calls for *Partition*. For a broadcast disk array of K disks, the number of procedure calls for *Partition* in algorithm VF^K is $K - 1$, whereas that in algorithm OPT is the number of internal nodes expanded by algorithm OPT in the solution tree. As will be shown in section 5, the solution obtained by algorithm VF^K is of high quality and is in fact very close to that of algorithm OPT. As the number of broadcast disks and the number of data items increase, the advantage of algorithm VF^K over algorithm OPT becomes more prominent.

5. Performance evaluation

In order to evaluate the performance of algorithm OPT and VF^K, we have implemented a simulation model of the broadcast environment. Specifically, the simulation model is described in section 5.1. Then, we examine the impact of employing hierarchical broadcast programs in section 5.2. Performance of algorithm VF^K and algorithm OPT is comparatively analyzed in section 5.3.

5.1. Simulation model

Table 5 summarizes the definitions for some primary simulation parameters. The number of data items to be broadcasted 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 [10]. It can be verified that the access frequencies become increasingly skewed as the value of θ increases. For comparison purposes, a scheme that evenly allocates data items to each broadcast disk, referred to as FLAT, is implemented. An example broadcast program generated by FLAT is the one shown in figure 1(a). Experimental results for algorithm VF^K, algorithm OPT and FLAT have been obtained and comparatively evaluated.

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 distribution parameter
FLAT	Scheme to generate a flat broadcast program

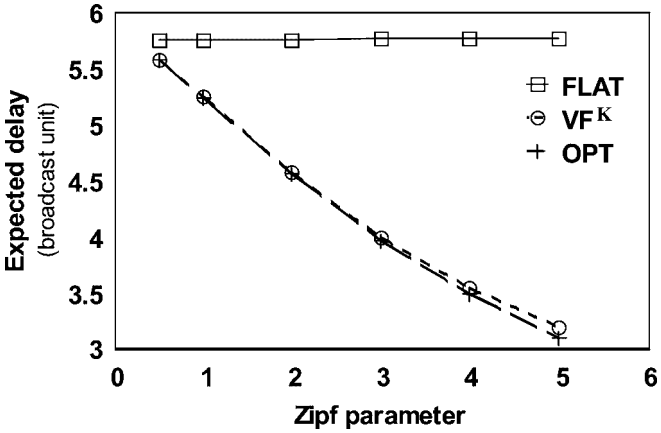

Figure 9. The average expected delay of FLAT, OPT and VF^K with the value of θ varied.

5.2. The impact of employing hierarchical broadcast programs

To show the advantage of generating hierarchical broadcast program, we set the value of n to 50 and the value of K to 4. The expected delay of data items under FLAT, OPT and VF^K 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 resulting expected delay of data items by running FLAT, OPT and VF^K are shown in figure 9. It can be seen from figure 9 that the access frequencies become increasingly skewed as the value of θ increases and the difference between expected delay of OPT and that of VF^K is almost negligeable, showing the very high quality of the solutions obtained by algorithm VF^K. Note that as the access frequencies become increasingly skewed, the hierarchical broadcast programs generated by OPT and VF^K perform significantly better than the flat broadcast program, indicating the very advantage of exploiting the feature of variant-fanout for the allocation tree generation in algorithm VF^K.

5.3. Comparative analysis for VF^K and OPT

We now examine the impact of varying the values of n and K to the performance of VF^K and OPT. First, in order to evaluate the impact of increasing the value of K, we set the value

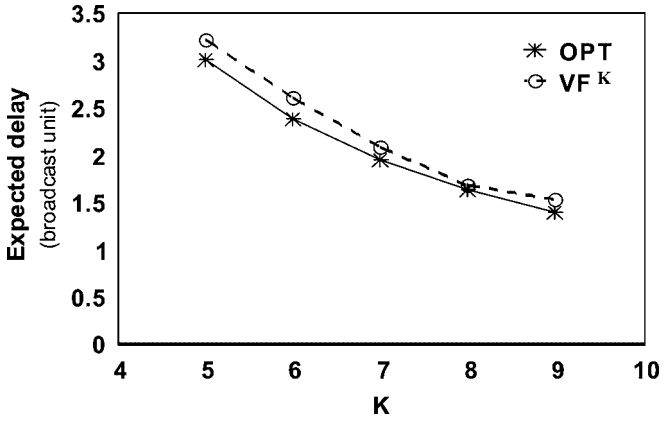

Figure 10. The expected delay of OPT and VF^K with the value of K varied.

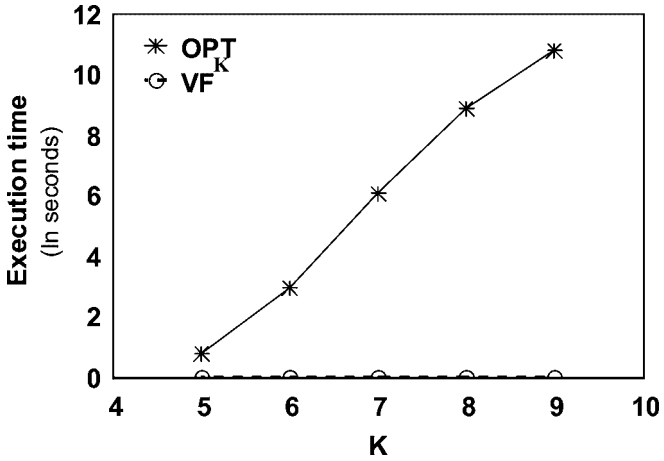

Figure 11. The execution time incurred by OPT and VF with the value of K varied.

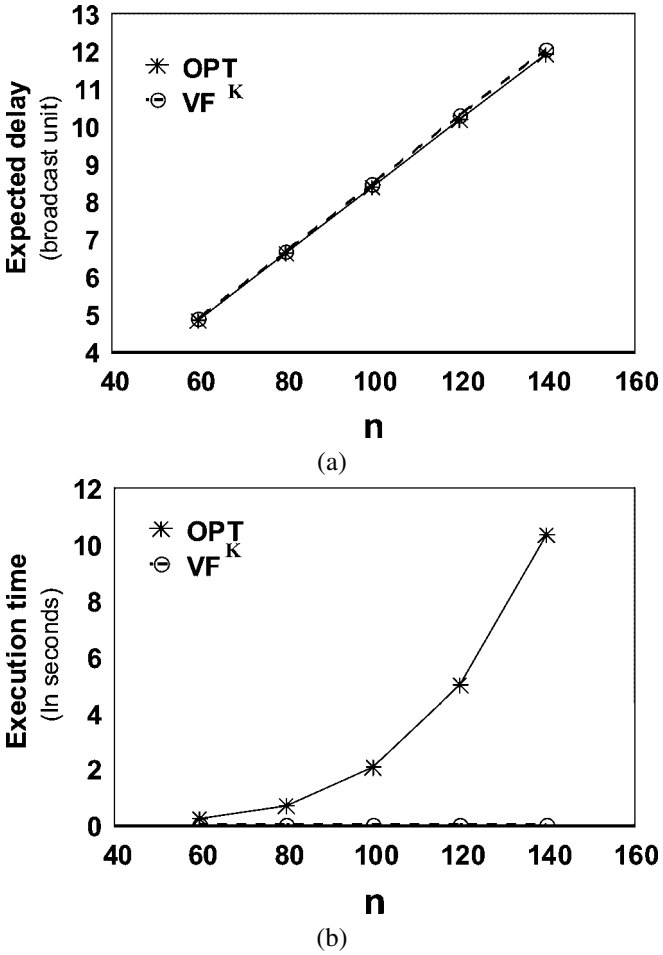

Figure 12. The performance comparison between OPT and VF^K with the value of n varied. (a) The expected delay of OPT and VF^K with the value of n varied. (b) The execution time incurred by OPT and VF^K with the value of n varied.

of n to 50 and the value of θ to 3. Figures 10 and 11 show the performance results of OPT and VF^K.

As can be seen in figure 10, the expected delay of OPT and VF^K decreases as the value of K increases. 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 VF^K and that of OPT is very small when the value of K varies, again showing the good quality and the robustness of solutions obtained by VF^K. From figure 11, the execution time incurred by OPT increases sharply as the value of K increases. This means that the advantage of algorithm VF^K over algorithm OPT becomes even more prominent when the number of K is large, since the expected delays resulted by the solutions from both algorithms are all very close to each other.

Next, the experiments of varying the value of n for OPT and VF^K are conducted where we set the value of K to 4 and the value of θ to 3. Figure 12 shows the performance comparison between OPT and VF^K when the value of n varies. In figure 12(a), as the number of data items to be broadcast increases, the expected delays of data items resulted by OPT and VF^K increase linearly as we anticipate. Also, the difference between the expected delays resulted by OPT and VF^K is negligible, again suggesting that algorithm VF^K be able to find a solution of very high quality efficiently when the value of n increases. Note that the execution time incurred by OPT is also significantly larger than that incurred by VF^K in figure 12(b), showing the increasing advantage of algorithm VF^K over algorithm OPT when the number of data items increases.

It is shown by our experimental results that algorithm VF^K is not only able to produce the solutions of very high quality but also of good scalability which is important for VF^K to be of practical use to generate hierarchical broadcast programs dynamically in a mobile computing environment.

6. Conclusions

In this paper, we explored the issue of generating hierarchical broadcast programs with the data access frequencies and the number of broadcast disks in a broadcast disk array given. Specifically, we first formulated the problem of generating hierarchical broadcast programs and transformed this problem into the one of constructing a channel allocation tree with

variant-fanout. By exploiting the feature of tree generation with variant-fanout, we developed a heuristic algorithm VF^K to minimize the expected delay of the corresponding broadcast program. In order to evaluate the solution quality obtained by algorithm VF^K and compare its resulting broadcast program with the optimal one, we devised an algorithm OPT based on a guide search to obtain the optimal solution. Performance of these algorithms was comparatively analyzed and sensitivity analysis on several parameters, including the number of data items and the number of broadcast disks, was conducted. It was shown by our simulation results that by exploiting the feature of variant-fanout in constructing the channel allocation tree, the solution obtained by algorithm VF^K is of very high quality and is in fact very close to the optimal one resulted by algorithm OPT. With its polynomial time complexity, algorithm VF^K incurs a much shorter execution time than algorithm OPT. As the number of broadcast disks and the number of data items increase, the advantage of algorithm VF^K over algorithm OPT becomes more prominent. By exploring the feature of variant-fanout tree construction, algorithm VF^K is not only able to produce the solutions of very high quality but also of good scalability which is important for algorithm VF^K to be of practical use to generate hierarchical broadcast programs dynamically in a mobile computing environment.

Appendix. Average expected delay of data items

The broadcast channel is divided into slots of same size that is equal to the size of data item. Assume that the number of data items to be broadcast is N_i and the size of slots for data items is one. Let $L_i(j)$ denote the number of slots from the end of slot j for data item i. Without loss of generality, the arrival requests for data items are triggered in the middle of slots. An illustrative example of a request for data item x while data item x is being pushed is shown in figure 13. Note that the latency of the request for data item x is equal to $1/2+L_x(x)$. Since the residual time, i.e., $1/2$, from the arrival point of the request until the end of the slot is independent of the program studied in this paper, we ignore it and use $L_x(x)$ as the measure of the expected delay. It can be verified that the $L_x(x)$ equals $N_i - 1$ in our illustrative example shown below. While the request of data item x may occur at any time slots, we can formulate the average expected delay of data item x as $\sum_{j=1}^{N_i} L_x(j)/N_i = \sum_{j=1}^{N_i}(N_i - j)/N_i$. For more detailed formulations, interested readers are referred to [1] and [23].

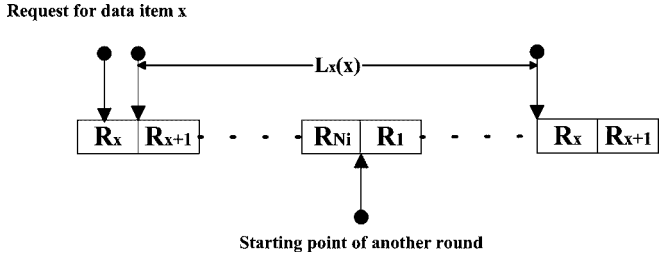

Figure 13. An illustrative example of the expected delay for data item x.

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] S. Acharya, M. Franklin and S. Zdonik, Balancing push and pull for data broadcast, in: *Proceedings of ACM SIGMOD* (May 1997) pp. 183–194.

[3] S. Acharya and S. Muthukrishnan, Scheduling on-demand broadcasts: new metrics and algorithms, in: *Proceedings of the 4th ACM/IEEE International Conference on Mobile Computing and Networking* (October 1998) pp. 43–54.

[4] A. Bar-Noy, J. Naor and B. Schieber, Pushing dependent data in Clients-Providers-Servers systems, in: *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking* (August 2000) pp. 222–230.

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

[6] 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.

[7] T.H. Gormen, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms* (MIT Press, Cambridge, MA, 1990).

[8] A. Datta, D.E. Vandermeer, A. Celik and V. Kumar, Broadcast protocols to support efficient retrieval from databases by mobile users, ACM Transactions on Database Systems, 24(l) (1999) 1–79.

[9] M.H. Dunham, Mobile computing and databases, in: *International Conference on Data Engineering* (February 1998), tutorial.

[10] 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.

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

[12] Q. Hu, W.-C. Lee and D.L. Lee, A hybrid indexing technique for power conserving wireless data broadcast, Journal on Distributed and Parallel Databases 9(2) (2001) 151–177.

[13] 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.

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

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

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

[17] W.-C. Lee and D.-L. Lee, Signature caching techniques for information filtering in mobile enviroments, Wireless Networks 5(1) (1999) 57–67.

[18] 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.

[19] N.J. Nilsson, *Principles of Artificial Intelligence* (Springer, Berlin, 1982).

[20] 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.

[21] N. Shivakumar and S. Venkatasubramanian, Energy efficient indexing for information dissemination in wireless systems, Wireless Networks and Applications 1(4) (1996) 433–446.

[22] 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.

[23] 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.

[24] 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.
[25] N.H. Vaidya and S. Hameed, Scheduling data broadcast in: asymmetric communication environments, Wireless Networks 5(3) (1999) 171–182.
[26] WAP application in Nokia, `http://www.nokia.com/corporate/wap/future.html`.
[27] WAP application in Unwired Planet, Inc., `http://phone.com`.
[28] M.-H. Yang, L.-W. Chen, Y.-C. Tseng and J.-P. Sheu, A traveling salesman mobility model and its location tracking in PCS networks, in: *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems* (April 2001) pp. 517–524.

Wen-Chih Peng was born in Hsinchu, Taiwan, R.O.C in 1973. He received the BS and MS degrees from National Chiao Tung University, Taiwan, in 1995 and 1997, respectively, and the Ph.D. degree in electrical engineering from the University of National Taiwan University, Taiwan, R.O.C. in 2001. During his Ph.D. program, he was mainly involved in the projects related to mobile computing, telecommunication databases and data mining. His research interests include mobile computing, mobile data management, and data mining. He is a member of the Phi Tau Phi Scholastic Honor Society.
E-mail: wcpeng@arbor.ee.ntu.edu.tw

Ming-Syan Chen received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, and the M.S. and Ph.D. degrees in computer, information and control engineering from The University of Michigan, Ann Arbor, MI, USA, in 1985 and 1988, respectively. Dr. Chen is currently a Professor in Electrical Engineering Department, National Taiwan University, Taipei, Taiwan. He was a research staff member at IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA from 1988 to 1996. His research interests include database systems, data mining, mobile computing systems, and multimedia networking, and he has published more than 120 papers in his research areas. In addition to serving as program committee members in many conferences, Dr. Chen is an associate editor of IEEE Transactions on Knowledge and Data Engineering on data mining and parallel database areas from 1997 to 2001, an editor of Journal of Information Science and Engineering, a distinguished visitor of IEEE Computer Society for Asia–Pacific from 1998 to 2000, and program chair of PAKDD-02 (Pacific Area Knowledge Discovery and Data Mining), Program vice-Chair of VLDB-2002 (Very Large Data Bases), general chair of Real-Time Multimedia System Workshop in 2001, program chair of IEEE ICDCS Workshop on Knowledge Discovery and Data Mining in the World Wide Web in 2000, and program co-chair of the International Conference on Mobile Data Management in 2003, and also of International Computer Symposium on Computer Networks, Internet and Multimedia in 1998 and 2000. He was a keynote speaker on Web Data Mining in International Computer Congress in Hong Kong, 1999, a tutorial speaker on Web Data Mining in DASFAA-1999 and on Parallel Databases in the llth IEEE International Conference on Data Engineering in 1995 and also a guest co-editor for IEEE Transactions on Knowledge and Data Engineering on a special issue for Data Mining in December 1996. He holds, or has applied for, eighteen U.S. patents and seven ROC patents in the areas of data mining, Web applications, interactive video playout, video server design, and concurrency and coherency control protocols. He received the Outstanding Innovation Award from IBM Corporate in 1994 for his contribution to parallel transaction design and implementation for a major database product, and numerous awards for his research, teaching, inventions and patent applications. Dr. Chen is a senior member of IEEE and a member of ACM.
E-mail: mschen@cc.ee.ntu.edu.tw