Optimal Index and Data Allocation in Multiple Broadcast Channels

Shou-Chih Lo and Arbee L.P. Chen*

Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 300, R.O.C.
Email: alpchen@cs.nthu.edu.tw

ABSTRACT

The issue of data broadcast has received much attention in mobile computing. A periodic broadcast of frequently requested data can reduce the workload of the up-link channel and facilitate data access for the mobile user. Since the mobile units usually have limited battery capacity, the minimization of the access latency for the broadcast data is an important problem. The indexing and scheduling techniques on the broadcast data should be considered. In this paper we propose a solution to find the optimal index and data allocation, which minimizes the access latency for any number of broadcast channels. We represent all the possible allocations as a tree in which the optimal one is searched, and propose a pruning strategy based on some properties to greatly reduce the search space. Experiments are performed to show the effectiveness of the pruning strategy. Moreover, we propose two heuristics to solve the same problem when the size of the broadcast data is large.

1. INTRODUCTION

Rapid advances in computer software/hardware and wireless network technologies have led to the development of mobile computing. In this environment, mobile users retrieve information by a portable computer. How to disseminate information efficiently to a large volume of mobile users becomes a challenge. Both the response time and the power consumption of a portable computer have to be considered. In [Fra98], an overview of delivery mechanisms is given. Among these mechanisms, data broadcast allows users to retrieve data simultaneously with a cost independent of the number of users. Any user can retrieve the broadcast data by listening to the channel, which results in a certain degree of energy saving. The disadvantage of data broadcast is that data can only be accessed sequentially. The users need to wait for the required data to appear on the broadcast channel. Current research on data broadcast can be divided into three categories:

1. *Determining the data for broadcasting*: The volume of broadcast data influences the access latency.

* To whom all correspondence should be sent.

A small set of data items is preferred to be broadcast. Usually, only most frequently accessed data items will be broadcast. How to dynamically broadcast the most frequently accessed data is the main problem. A common technique is to drop a data item from the broadcast channel and re-estimate its access frequency from the on-demand data requests [DCK97, SRB97] to make sure that the dropped data item is actually no more frequently accessed.

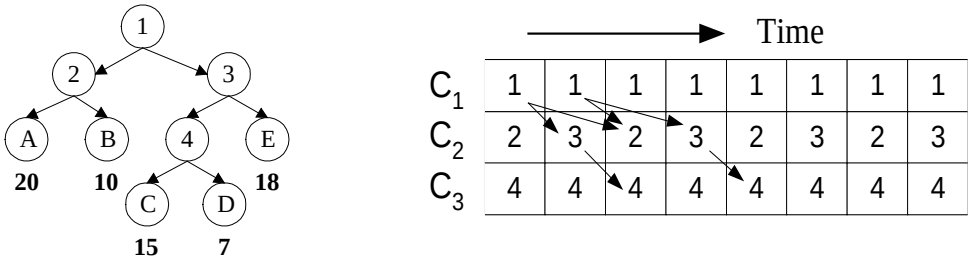
2. *Scheduling the data broadcasting*: The data items to be broadcast may have different access frequencies. Given the data access frequencies, the problem is how to arrange the broadcast data to minimize the average response time for data requests. The probabilistic and periodic broadcasting methods were studied in [IV94] and [Ach95, AFZ96], respectively. The issue of fault tolerance on the broadcast data was studied in [LS95, LC97, TO98]. In [TY98], the broadcast data are rearranged based on the structure proposed in [Ach95] for supporting range queries.

3. *Indexing the broadcast data*: Building indexes on the broadcast data helps users decide whether their desired data are in the broadcast channel and when they become available. During the time waiting for the data, the portable computer can be turned into a power saving mode. The problem is how to mix the index with the data items such that the response time and the number of index accesses on the broadcast channel can be minimized. Index techniques such as distributed index, hashing and signature were studied in [IVB94a, IVB94b, IVB94c, LL96, TY96, TY97].

Imielinski et al. [IVB94a, IVB94b] propose an index technique on the broadcast data, where each data item is considered to have the same access frequency. The *access time* (the time elapsed from the moment a user poses a request to the moment the result is downloaded by the user) and *tuning time* (the time spent by the user listening to the channel) are used to estimate the response time and the power consumption for a data request. If each data item has different access frequencies, two possible approaches can be adopted. One is to broadcast the popular data more often, which minimizes the average access time [IV94, Ach95]. The index techniques for this approach are discussed in [TY97], which are variations of the work done in [IVB94a].

The other is to construct a skewed index tree with the popular data having a shorter path from the root of the index tree, which minimizes the average tuning time [CYW97, SV96]. The construction of this index tree is analogous to that of Huffman code, which is pointed out in [SV96]. However, the users may fail to find a desired data item by traversing the Huffman tree thus constructed, given the key of the desired data item. In [CYW97], the proposed algorithms for constructing the skewed index tree have the same problem. Another class of Huffman trees termed *Alphabetic Huffman tree* is proposed in

[HT71], which functions as a binary search tree. It is extended to k-nary search trees in [SV96] such that by adjusting the fanout of the tree, a tree node can fit in a wireless packet of any size. In this paper, we consider this k-nary search tree as our index tree and discuss the problem of index and data allocation in multiple broadcast channels for minimizing both the average access time and the average tuning time.



(a) Index Tree (b) Corresponding Index Allocation

Fig. 1. Example of Index Allocation.

1.1. Motivation

An example index tree with fanout 2 is shown in Fig. 1(a), where nodes with numeric labels are index nodes, nodes with alphabetic labels are data nodes, and each data node is associated with an access frequency. Conventional techniques discussed in [IVB94a, IVB94c] focus on index and data allocation in a single broadcast channel. The issue of multiple broadcast channels is addressed in [SV96].

In [SV96], the index and data nodes are allocated to separate channels and only the allocation of index nodes is discussed. Fig. 1(b) shows the corresponding allocation of index nodes, where C_i denotes the channel number. Each index node allocated in the channels has pointers to the nearest index nodes allocated in the subsequent channel, which correspond to its children in the index tree. Also, the index nodes at the last channel have pointers to the indexed data in other channels. The index nodes on each level of the tree are assigned to an individual channel, and transmitted cyclically. The user can tune into any place in the first channel to get the root of the index tree and follow the index pointers to retrieve the desired data item.

This kind of index allocation has two problems:

- Lack of flexibility: The number of needed channels to allocate the index nodes is fixed, which is equal to the depth of the index tree (excluding the data nodes). However, there may not exist sufficient channels for the allocation. Although we can change the fanout of the index tree such that all index nodes can be allocated to the available channels, the index tree cannot show the effect of

the skewness for accessing data with different access frequencies.

- Waste of channel space: Take an extreme case where the index tree is a chain as an example. Allocating each index node in an individual channel presents space waste. One channel is enough, since any two of the index nodes can not be accessed simultaneously.

In this paper, we propose a parameterized scheduling algorithm which avoids the above drawbacks, given the available number of broadcast channels and an index tree. In our approach, the index and data nodes are mixed into the same channels for an easy measurement of the access time. We map the problem into the *Personnel Assignment Problem* [Str89] from which we derive the techniques to solve the problem.

The rest of this paper is organized as follows. In Section 2, we formalize the problem of index and data allocation. In Section 3, we present a tree representation for all possible allocations. Some properties for pruning the tree space are also introduced. Section 4 discusses the performance of the pruning techniques and introduces two heuristics to deal with the situation when the size of the broadcast data is large. Finally, we present the conclusion and future work in Section 5.

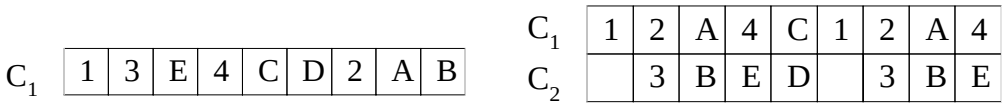
2. PROBLEM DESCRIPTION

In this section, we first introduce the structure of the broadcast and then describe the problem of index and data allocation.

2.1. Preliminaries

Each periodic broadcast constitutes a *broadcast cycle*. The server will adjust the content of each broadcast cycle to satisfy the current needs from clients. Consider an index tree composed of index nodes and data nodes. The size of an index node is assumed equal to the size of a data node. The logical unit of a broadcast is bucket. Each bucket in the broadcast can accommodate an index node or a data node. The user has to first tune into the first broadcast channel to get the bucket containing the root of the index tree and then continues the data access. The pointer data in each index node are represented by the <*CNUM*, *OFFSET*> pair, indicating the channel number and the offset in number of buckets for retrieving the next relevant bucket. A portable computer can only access a single channel at any time instance. The time for switching from one channel to another is assumed negligible. The access time can be further divided into two parts: *probe wait* and *data wait*. The probe wait is the time to get the bucket containing the root of the index tree and the data wait is the time after the probe wait

to the moment the required data item is retrieved. In this study, we focus on the data wait which is measured by the unit of bucket.



(a) One Broadcast Channel (b) Two Broadcast Channels

Fig. 2. Two Possible Index and Data Allocation.

2.2. Problem Formulation

In this subsection, a formal description of the problem is given. Also, we introduce another similar problem for deriving the problem solving techniques.

The Index and Data Allocation Problem: We are given multiple broadcast channels and an index tree. Let the broadcast channels be $C = \{C_1, C_2, C_3, ..., C_k\}$, the set of index nodes in the index tree be $I = \{I_1, I_2, I_3, ..., I_m\}$, and the set of data nodes in the index tree be $D = \{D_1, D_2, D_3, ..., D_n\}$. Each data node D_i is associated with a weight $W(D_i)$ indicating the average data access frequency for the node. Assume one bucket is transmitted at one slot of the broadcast channel in one time unit. Then, an index and data allocation can be represented as a mapping function $f : I \cup D \rightarrow C \times S$, where S denotes the domain of channel slots. To simplify the problem, we assume that no index or data nodes replicate in a broadcast cycle. In other words, the function f can be viewed as a one-to-one function in a broadcast cycle. A feasible allocation has to satisfy the condition in which a child node should be broadcast after its parent node in the index tree. Fig. 2 presents two such allocations for the index tree shown in Fig. 1(a). To capture the root of the index tree for any user tuning into the broadcast channel, all buckets in the first broadcast channel have a pointer to the first bucket of the next broadcast cycle. Let $T(D_i)$ denote the offset in terms of channel slots between the first bucket of a broadcast cycle and data node D_i. The average data wait for accessing the data nodes can be obtained by

$$\frac{\sum_{D_i \in D} W(D_i) \cdot T(D_i)}{\sum_{D_i \in D} W(D_i)} \qquad (1)$$

For example, the data waits for the allocations shown in Fig. 2 are as follows.

(i) $\frac{1}{70}(18 \times 3 + 15 \times 5 + 7 \times 6 + 20 \times 8 + 10 \times 9) = 6.01$ for the one channel case.

(ii) $\frac{1}{70}(20 \times 3 + 10 \times 3 + 18 \times 4 + 15 \times 5 + 7 \times 5) = 3.88$ for the two channel case.

Our goal is to find an optimal allocation which minimizes formula 1. Notice that there may exist more than one optimal allocation for a given index tree.

Before discussing how the problem can be solved, we introduce another similar problem called *Personnel Assignment Problem* in the following.

The Personnel Assignment Problem: Given a linearly ordered set of persons $P = \{P_1, P_2, P_3, ..., P_n\}$, where $P_1 < P_2 < ... < P_n$ and a partially ordered set of jobs $J = \{J_1, J_2, J_3, ..., J_n\}$. A personnel assignment can be represented as a mapping function $f : J \rightarrow P$. The function f is a one-to-one function, which means each person is assigned only one job and vice versa. A feasible assignment has to satisfy the condition: if $J_i \leq J_j$ then $f(J_i) < f(J_j)$. Let C_{ij} denote the cost of assigning J_i to P_j, and X_{ij} be 1 if J_i is assigned to P_j and 0 otherwise. Then the problem can be stated as to find an optimal assignment which minimizes the cost $\sum_{J_i \in J \text{ and } P_j \in P} C_{ij} X_{ij}$. This problem has been proved to be NP-hard.

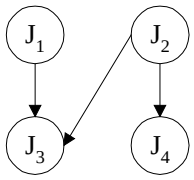
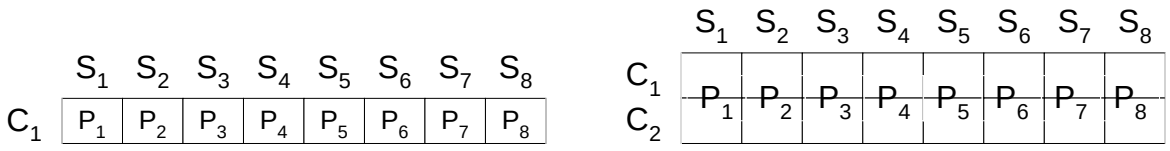


Fig. 3. A Partial Ordering of Jobs.

Consider the example shown in Fig. 3. Here, $P = \{P_1, P_2, P_3, P_4\}$, $J = \{J_1, J_2, J_3, J_4\}$, and the partial ordering of the jobs is $J_1 \leq J_3$, $J_2 \leq J_4$ and $J_2 \leq J_3$. In this case, $J_1 \rightarrow P_1$, $J_2 \rightarrow P_2$, $J_3 \rightarrow P_3$ and $J_4 \rightarrow P_4$ is a feasible assignment. In the following, we show that the index and data allocation problem can be transformed to the personnel assignment problem.



(a) For One Broadcast Channel (b) For Two Broadcast Channels

Fig. 4. The Mapping between Persons and Channel Slots.

Problem Transformation: The behavior of allocating the index and data at channel slots is equivalent to that of assigning jobs to persons. The index and data nodes can be regarded as jobs (i.e., $J = I \cup D$). The index tree holds the ordering relationship of the jobs. On the other hand, the channel

slots are regarded as persons (i.e., $P = S$). The channel slots inherently hold a linear ordering relationship. For a single broadcast channel, the mapping of a channel slot to a person is depicted in Fig. 4(a). Consider multiple broadcast channels. Among the buckets transmitted at the same slots of different channels, only one bucket at a broadcast channel can be accessed. Therefore, we should put as many index or data nodes without the ordering relationship at the same channel slots as possible for shortening the broadcast cycle. This means for the multiple broadcast channel case, each channel slot is also mapped to a person. More than one job with no ordering relationship can be assigned to a person while only one job can be performed at a time. The mapping of channel slots to persons for the two broadcast channel case is shown in Fig. 4(b).

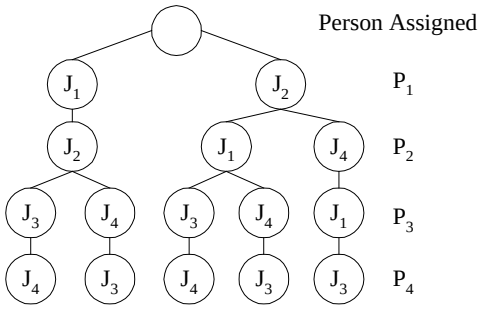


Fig. 5. A Tree Representation of All the Possible Assignments.

3. THE PRUNING ALGORITHM

In this section, we introduce a tree structure to represent all the possible index and data allocations and describe the pruning techniques to reduce the search space for the optimal allocation.

3.1. Solution Space Representation

In the personnel assignment problem, it can be shown that all of the feasible assignments can be generated by the topological sorting of the jobs. The topological sorting holds the property that if *i* is a predecessor of *j* in the partial ordering relationship then *i* is located before *j* in the sorted sequence. This just satisfies the condition of the assignment problem. All these sorting sequences can be represented by a tree structure (called *topological tree* in this paper) as shown in Fig. 5. Each path from the root to the leaf of the topological tree represents a feasible assignment. As a result, the assignment problem can now be viewed as one to find a path with the minimum cost (called *optimal path*) among all these paths in the topological tree.

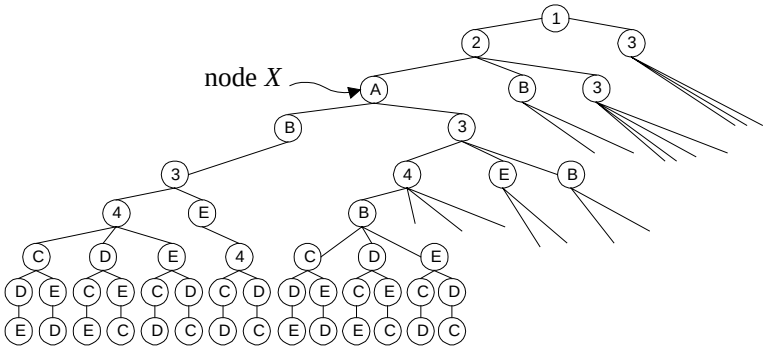


Fig. 6. The 1-Channel Topological Tree.

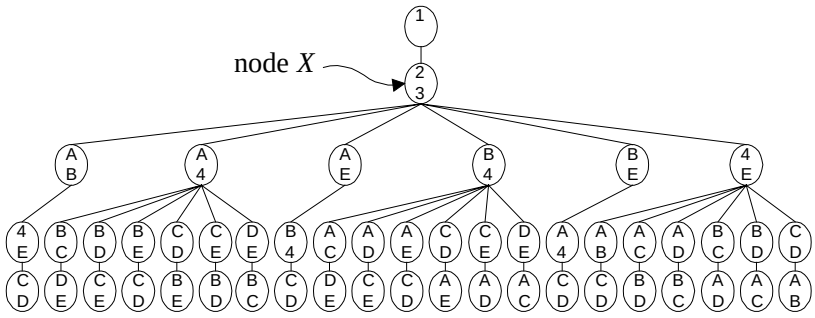


Fig. 7. The 2-Channel Topological Tree.

Similarly, we can represent all the possible index and data allocations in our problem as a topological tree. Take the tree of Fig. 1(a) as an example. The topological trees for one and two broadcast channels are presented in Fig. 6 and Fig. 7, respectively. For multiple channels, each node in the topological tree is a compound node which contains the buckets transmitted at the same slots of the channels. Each bucket in a compound node is called *element* which may be an index node or a data node. Below, the algorithm to construct a topological tree for *k* broadcast channels is given.

Algorithm 1 (Generating a *k*-channel topological tree)

Input. An index tree with data items on the leaf nodes.

Output. A *k*-channel topological tree.

Begin

1) Create the root of the topological tree, which contains the root of the index tree.
2) For each leaf node *P* of the topological tree
3) Construct a reduced graph from the index tree by removing the vertices which belong to the nodes along the path from the root to *P* in the topological tree, and all edges leading out of these vertices.
4) Collect the vertices that have no predecessor in the reduced graph as a set *S*. If the number of

vertices in S is no larger than k, then create a node containing all the vertices in S as a child of P. Otherwise, for each k-component subset of S, create a corresponding node as a child of P.

5) End For

6) If the reduced graph in each iteration on the above loop is empty then stop, otherwise go to Step 2).

End.

In general, an optimal path in a k-channel topological tree can be found by using the best-first search strategy. Here, we give a simple evaluation function $E(X)$ for the search. Define $E(X)$ as $E(X) = V(X) + U(X)$, where $V(X)$ is the data wait for the data nodes along the path from the root to the node X in the topological tree, and $U(X)$ is the estimated data wait for other data nodes not in $V(X)$. $V(X)$ can be cumulatively computed in each step of the search, while $U(X)$ is acquired by assuming the data nodes associated with $U(X)$ are all allocated next to the node X.

The broadcast is then generated by allocating the elements of each node from the top of the optimal path at separate slots of the channels. Elements in a node of the topological tree can be allocated at the same slots of different channels level by level. In order to reduce the number of switches between channels when accessing data, the following rules are enforced:

- Put the element in the root node into the first broadcast channel.
- Put the elements of nodes which have the parent-child relationship in the index tree into the same broadcast channel if possible.

3.2. TREE PRUNING TECHNIQUES

As can be seen in Fig. 6, the constructed topological tree may be huge. In this subsection, we introduce the techniques to prune the tree space. For convenience of explanation, we call the parent and the child of a node in the topological tree the *last-neighbor* and *next-neighbor* of the node, respectively, while use the terms children, parents and ancestors for the index tree. Consider the set of the next-neighbors for a node in the topological tree as our candidate set. Our goal is to derive some properties for eliminating the elements in the candidate set, which are apparently not included in the optimal path. At first, we use the following notations to explain how to generate all the next-neighbors for a node X in a k-channel topological tree.

DEFINITION.

$PATH_T(X)$: The set of elements in the nodes along the path from the root to the node X in the topological tree.

Children(x): The set of children for the node x in the index tree.

Neighbor$_k$(X): The set of next-neighbors for the node X in a k-channel topological tree.

Denote the elements in *Neighbor*$_k$(X) as S. It can be observed from Algorithm 1 that S is first generated by collecting the set of children in the index tree for the elements in $PATH_T(X)$, and then eliminating the elements in $PATH_T(X)$ from S. That is, $S = \bigcup_{y \in PATH_T(X)} Children(y) - PATH_T(X)$. Each next-neighbor is generated by each k-component subset of S. Let P_k^S denote the set of all the k-component subsets of S. Then,

$$Neighbor_k(X) = \begin{cases} P_{|S|}^S, \text{ if } k \geq |S| \\ P_k^S, \text{ if } k < |S| \end{cases}, \text{ where } |S| \text{ is the number of elements in } S.$$

EXAMPLE 1. In Fig. 6, $PATH_T(X) = \{1,2,A\}$, $\bigcup_{y \in PATH_T(X)} Children(y) = \{2,3,A,B\}$, $S = \{3,B\}$, and *Neighbor*$_1$(X)={{3},{B}}. In Fig. 7, $PATH_T(X) = \{1,2,3\}$, $\bigcup_{y \in PATH_T(X)} Children(y) = \{2,3,4,A,B,E\}$, S = {4,A,B,E}, and *Neighbor*$_2$(X)={{A,4},{B,4},{4,E},{A,B},{A,E},{B,E}}.

The rationale of the pruning technique is to swap each next-neighbor in *Neighbor*$_k$(X) with the node X and to delete the next-neighbor with its following nodes in the topological tree, if the corresponding path has better access cost after swapping. There are two kinds of swaps: *global swap* and *local swap*. In the global swap, the positions of two nodes are exchanged, while part of the elements between the two nodes are exchanged in the local swap. To confirm a unidirectional exchange of two elements which are both index nodes in the local swap, each index node in the index tree is also associated with a unique weight. The weight can be given by numbering the index nodes from 1 by the preorder traversal of the index tree. Let the two nodes to be swapped be X and Y (Y is a next-neighbor of X). Assume the elements in both nodes are $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_m\}$, where $1 \leq n, m \leq k$.

We have the following properties.

LEMMA 1. The nodes X and Y can be globally swapped, if there is no parent-child relationship between all the elements in X and Y.

LEMMA 2. If nodes X and Y can be globally swapped, it is beneficial to allocate X before Y only when $\sum_{\substack{x_i \in X \text{ and} \\ x_i \text{ is a data node}}} W(x_i) \geq \sum_{\substack{y_i \in Y \text{ and} \\ y_i \text{ is a data node}}} W(y_i)$.

LEMMA 3. For the data nodes having the same parent in the index tree, their positions in the path of the topological tree can be better arranged by the descending order of their access frequencies.

PROPERTY 1. If all the index nodes have been allocated, the best arrangement for the broadcast is to allocate the remaining data nodes in descending order of their access frequencies.

EXAMPLE 2. In Fig. 6, the path with the subsequence *ECD* is the best one among the leftmost six paths. In Fig. 7, the path with the subsequence *CEBD* is the best one among the paths containing the node *A4* at the third level.

PROPERTY 2. Consider a node X in a 1-channel topological tree. We can restrain the next-neighbors of X to have the following characteristics:

1. If the element in X is an index node x, then the next-neighbors of X are either index nodes which are the children of x or the data node which is the child of x with the largest weight. Assume x has n children, denoted by $Children(x) = \{I_1, I_2, \ldots, I_k, D_1, D_2, \ldots, D_{n-k}\}$, where I_i and D_i represent an index node and a data node, respectively. If $W(D_1) \geq W(D_2) \geq \ldots \geq W(D_{n-k})$, then $Neighbor_1(X) = \{\{I_1\}, \{I_2\}, \ldots, \{I_k\}, \{D_1\}\}$.
2. If the element in X is a data node x, then the next-neighbors will not contain a data node with its weight larger than that of x.

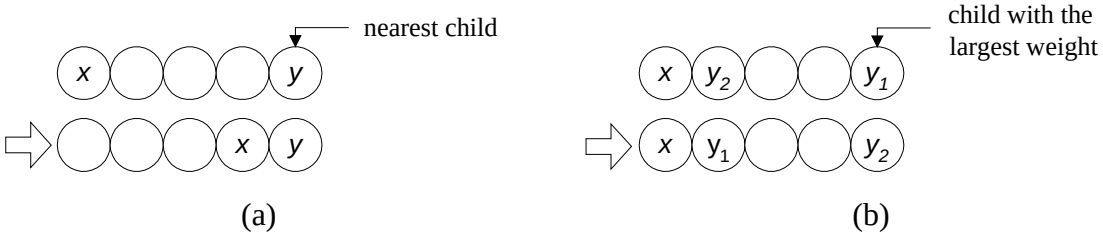


Fig. 8. Reordering the Broadcast.

PROOF. At first, we prove that an index node x can be always adjacent to one of its children in the path of a topological tree. If this condition is violated, we can move x to be beside the nearest child y as shown in Fig. 8(a) according to Lemma 1. The new broadcast allocation is certainly better than the old one because the nodes between x and y are all shifted one left and the nodes behind y keep at the same places. Second, we justify that if the next-neighbor is a data node then the data node has the largest weight among the children of x. According to Lemma 3, we can rearrange the relative positions of these data nodes as shown in Fig. 8(b) for getting a better broadcast allocation. Finally, we can confirm that two neighboring data nodes x and y should be well allocated with x before y as $W(x) \geq W(y)$

according to Lemma 2.

EXAMPLE 3. Consider the tree in Fig. 6. Among the next-neighbors *A*, *B*, and *3* of the node containing index node 2 at the second level, only the node *A* remains (by the first characteristic) after the pruning. Also, the leftmost path can be eliminated due to the subsequence *DE* where $W(D) < W(E)$ (by the second characteristic).

The topological tree in Fig. 6 can be reduced to the one shown in Fig. 9 by Property 1 and Property 2.

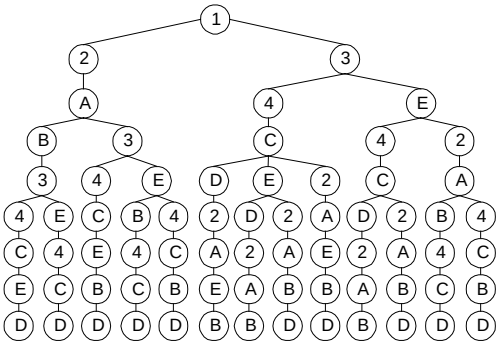


Fig. 9. The 1-channel Topological Tree after Pruning.

LEMMA 4. The nodes *X* and *Y* can be locally swapped if there are at least two elements x_i in *X* and y_i in *Y* such that the children of x_i are not in *Y* and y_i is not a child of any element in *X*.

LEMMA 5. If the elements of *X* are all index nodes and there is a y_i in *Y* such that y_i is not a child of any element in *X*, then *X* can be locally swapped with *Y*.

PROOF. Assume there are *n* elements in *X* and *m* elements in *Y*. If $n < m$ then y_i can be moved into node *X*. If $n \geq m$ it has to exist one element x_i in *X* such that the children of x_i are not in *Y*. The reason is that the number of parents is less than that of children. Therefore, x_i can be swapped with y_i.

PROPERTY 3. Consider a node *X* in a *k*-channel ($k > 1$) topological tree. We can restrain the next-neighbors of *X* to have the following characteristics:

1. If the elements in *X* are all index nodes, there is at least one element in the next-neighbor of *X*, which is a child of an element in *X*. Moreover, every data node in the next-neighbor is a child of an element in *X*.

2. If there are *n* data nodes in the next-neighbor of *X*, which are also the children of the elements in *X*, these data nodes must be the first *n* large ones among all the children of the elements in *X* in terms

of their access frequencies.

3. There is no index node y_i in the next-neighbor of X, which can be locally swapped with an index node x_i in X and $W(y_i) > W(x_i)$.

4. Every data node in the next-neighbor of X must be a child of an element in X if its weight is larger than that of a data node in X.

PROOF. For the first characteristic, the global swap can be performed if X is not adjacent to one of the children of the element in X, and the local swap can be performed according to Lemma 5 if there is such a data node in the next-neighbor. For the second characteristic, we can reorder these data nodes to satisfy this condition according to Lemma 3. The third characteristic confirms the unidirectional exchange between two index nodes. The local swap can guarantee the fourth characteristic.

EXAMPLE 4. Consider the tree in Fig. 8. All paths having the node *B4* at the third level are worse than those having the node *A4* at the third level (by the second characteristic). The leftmost path can be eliminated due to the subsequence *AB4E* where $W(E) > W(B)$ (by the fourth characteristic).

The topological tree of Fig. 7 can be reduced to the one shown in Fig. 10 by Property 1 and Property 3. A general algorithm to create a reduced *k*-channel topological tree is given in Appendix.

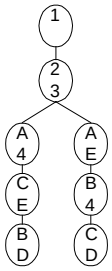


Fig. 10. The 2-channel Topological Tree after Pruning.

COROLLARY 1. If the number of broadcast channels is larger than the maximal number of nodes at the same level of an index tree, the optimal allocation is to assign the nodes at the same level into the same slots of different channels.

3.3. Pruning the Data Tree

In this subsection, we introduce another technique to further prune the 1-channel topological tree. Remember that the data nodes are mainly concerned in the problem of index and data allocation. The index nodes are ignored in evaluating the access cost. This motivates us to construct a similar tree by considering only the data nodes. This kind of tree is called *data tree*. We will present the construction

and the pruning of the data tree in the following.

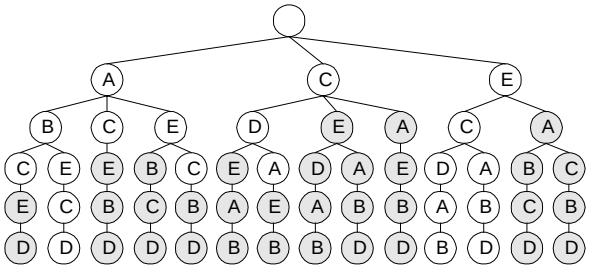


Fig. 11. An Example of a Data Tree.

The data tree is derived from the topological tree as below. For a data node, its previous data node in the path of the topological tree becomes its parent in the data tree, while each following data node in the path becomes its child. An example of a data tree derived from Fig. 9 is shown in Fig. 11. For a path from the root to the leaf of a data tree, let D_1 denote the first data node and D_i denote the ith data node. To produce a broadcast from a data tree, the information about index nodes is recorded in each data node. For a data node D_i, two types of index information *Cancestor*(D_i) and *Nancestor*(D_i) are recorded. Their respective meanings are defined below.

DEFINITION.

$PATH_D(D_i)$: The set of data nodes along the path from the root to the data node D_i in the data tree.
Ancestor(D_i): The set of ancestors of the data node D_i in the index tree.
Cancestor(D_i): The set of ancestors of the data nodes in $PATH_D(D_i)$ (see equation (i)).
Nancestor(D_i): The set of ancestors of the data node D_i, which should be allocated among D_{i-1} and D_i (see equation (ii)).

(i) $Cancestor(D_i) = \bigcup_{Y \in PATH_D(D_i)} Ancestor(Y)$.

(ii) $Nancestor(D_i) = Ancestor(D_i) - Cancestor(D_{i-1})$.

An example of a partial data tree with index information is shown in Fig. 12, where each node is associated with a *Nancestor* and *Cancestor* pair. A broadcast can be generated from one path of the data tree by the following procedure.

For i = 1 to $|D|$ ($|D|$ *is the number of all data nodes*)
    Output the associated index nodes in Nancestor(D_i) *and then output the data node* D_i.
End For

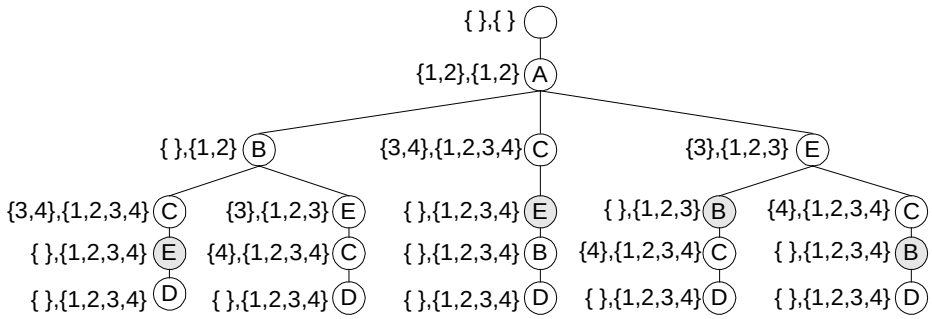


Fig. 12. Pruning the Data Tree.

Consider the leftmost path in Fig. 12, the generated broadcast is *12AB34CED*. Our goal is still to find the optimal path in the data tree. Here, we use the same pruning technique of swapping two neighboring nodes in the data tree to reduce the tree space. As mentioned before, the index nodes are recorded in the node of the data tree. We should take these index nodes into account when performing swapping. Consider the leftmost path in Fig. 12 again. For the two data nodes *C* and *E*, the corresponding subsequences in the broadcast are *34C* and *E*, respectively. However, we cannot directly exchange the two subsequences. Since the index node *3* is also the parent of *E*, it should not be involved in the exchange. That is, the common ancestors should be excluded in the exchange of two data nodes. Consider two neighboring data nodes D_i and D_{i+1} in the path of a data tree. The corresponding exchangeable subsequences are as follows.

subsequence of D_i: *the associated index nodes recorded in* $(Nancestor(D_i) - Ancestor(D_{i+1}))$ *followed by the data node* D_i.

subsequence of D_{i+1}: *the associated index nodes recorded in* $Nancestor(D_{i+1})$ *followed by the data node* D_{i+1}.

We use the following lemma to decide the better order for the two exchangeable subsequences.

LEMMA 6. Consider two exchangeable subsequences *A* and *B*, where *A* has N_A nodes and *B* has N_B nodes. If the condition $N_B \cdot \sum_{x \in A \text{ and } x \text{ is a data node}} W(x) \geq N_A \cdot \sum_{y \in B \text{ and } y \text{ is a data node}} W(y)$ is satisfied, then the arrangement *AB* is better than *BA*.

PROOF. Assume there are *n* data nodes in subsequence *A* and *m* data nodes in subsequence *B*. For a data node in a subsequence, we use the (W_i, P_i) pair to denote its access frequency and its position in the subsequence. Then, we have the following information.

for subsequence *A*: $(W_1, P_1), (W_2, P_2), \ldots, (W_n, P_n)$, $1 \leq n \leq N_A$, $1 \leq P_i \leq N_A$

for subsequence B: (W'_1, P'_1), (W'_2, P'_2), …, (W'_m, P'_m), $1 \le m \le N_B$, $1 \le P'_i \le N_B$

The data waits for the orders AB and BA are calculated by the formulas.

$$(AB) \quad \sum_{i=1}^{n} W_i P_i + \sum_{i=1}^{m} W'_i (P'_i + N_A)$$

$$(BA) \quad \sum_{i=1}^{m} W'_i P'_i + \sum_{i=1}^{n} W_i (P_i + N_B)$$

$$(AB–BA) \quad N_A \sum_{i=1}^{m} W'_i - N_B \sum_{i=1}^{n} W_i$$

PROPERTY 4. Consider two neighboring data nodes D_i and D_{i+1} in the path of a data tree. It will be better to allocate D_i before D_{i+1}, if the following condition is satisfied.

$$\left(\left|Nancestor(D_{i+1})\right| + 1\right) \cdot W(D_i) \ge \left(\left|Nancestor(D_i) - Ancestor(D_{i+1})\right| + 1\right) \cdot W(D_{i+1})$$

Consider the previous example with data nodes C and E. Since $Nancestor(C) = \{3,4\}$, $Nancestor(E) = \{\}$, and $Ancestor(E) = \{1,3\}$, the exchangeable subsequences are $4C$ and E. Given $W(C) = 15$ and $W(E) = 18$, the condition $1 \times 15 \ge 2 \times 18$ is not satisfied in Property 4. Therefore, this path is not an optimal one and can be pruned. We mark the nodes that violate the condition after adding these nodes into the data tree in Fig. 12. In Fig. 11, we mark all the nodes that will not be generated in the final data tree according to Property 4. Only three paths remain in the final data tree.

CORALLARY 2. The pruning technique illustrated in Property 4 can be extended from the one-and-one exchange to m-and-n ($m, n > 1$) exchange.

4. THE HEURISTIC ALGORITHM

In this section, we first discuss the pruning effect on the index and data allocation problem. Then, we present two heuristics to deal with the same problem when the size of the broadcast data is large.

4.1. Performance of Pruning

As can be seen, the original topological tree before pruning can be huge especially for a single broadcast channel. In the following, we give an analysis on the performance of pruning for a single broadcast channel, where the data tree is applied.

As mentioned in Property 2, an index node has to be adjacent to one of its children. This reflects

that there is only one way to insert the index nodes into a sequence of data nodes. Therefore, we can view the index and data allocation as the permutation of only data nodes. Let the data nodes having the same parent in an index tree constitute a group. Suppose there are totally n groups each with an average m nodes. Since the data nodes in the same group have only one order (i.e., the descending order of their access frequencies) in the broadcast allocation, the number of the allocations limited by Property 2 equals that of the permutations of n kinds of objects each with m objects, which is $\frac{(nm)!}{(m!)^n}$. For Property 1, the following phenomenon is reflected. Once a data node in each group has been allocated, the remaining data nodes have only one order in the allocation. There is no simple closed-form expression for this effect. Similarly, there is no simple closed-form expression for the pruning effect stated in Property 4. We do an experiment on a full balanced m-nary tree to show these pruning effects. The access frequency of each data node is given randomly. The result is shown in Table 1, where we investigate the total number of paths from the root to the leaves in the reduced data tree and show the percentage of pruning $(1-\frac{Total\ Paths}{(m^2)!})$. It can be seen that the performance degrades as the index tree is getting larger, which means the pruning algorithm is applicable only to a small size of the problem.

Index Tree (The depth is 3)	By Property 2		By Property 1, 2		By Property 1, 2, 4	
	Total Paths	Pruning %	Total Paths	Pruning %	Total Paths	Pruning %
m = 2	6	75%	4	83.33%	1	95.83%
m = 3	1680	99.5%	186	99.95%	3	99.99%
m = 4	6306300	99.99%	438048	99.99%	16	99.99%
m = 5	$\approx 6.2\times10^{14}$	99.99%	N/A	N/A	464	99.99%
m = 6	$\approx 2.7\times10^{24}$	99.99%	N/A	N/A	1366361	99.99%

Table 1. Pruning Effects.

4.2. Heuristic Approaches

For dealing with a larger size of the problem, other techniques are required. Here, we give two heuristics.

1) Index Tree Shrinking. We first reduce the size of the index tree by the following two ways and then find the optimal path from the reduced index tree.

Node combination. We change the index node whose children are all data nodes into a data node having the weight equal to the sum of the weights of the children. This is repeated until that the index

tree becomes small enough to be able to efficiently find an optimal path. Once an optimal path is found, we restore the combined nodes in the optimal path to produce the broadcast.

Tree partitioning. We divide an index tree into several subtrees and find an optimal path for each one. Then, we merge these paths to produce the broadcast.

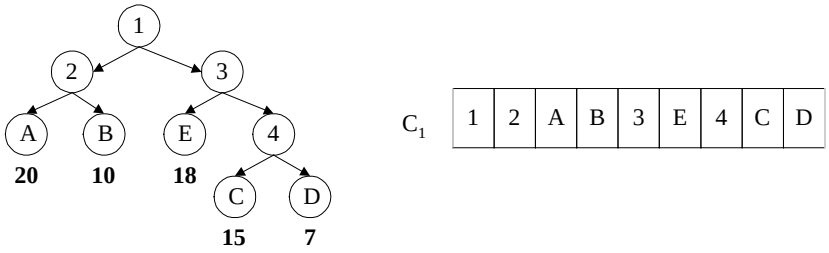


Fig. 13. Index Tree Sorting and the Single Channel Allocation.

2) Index Tree Sorting. For each node in the index tree, we sort its children from left to right in descending order denoted by $\triangleright$. Given two nodes *A* and *B* in the index tree, assume there are N_A and N_B nodes in the subtrees rooted at *A* and *B*, respectively.

$$A \triangleright B \text{ if } N_B \cdot \sum_{x \in Subtree(A) \text{ and } x \text{ is a data node}} W(x) \geq N_A \cdot \sum_{y \in Subtree(B) \text{ and } y \text{ is a data node}} W(y).$$

After we sort all the nodes, we traverse the index tree in preorder to produce the broadcast. For example, we sort the pairs of the nodes *23*, *AB*, *4E* and *CD* in Fig. 1(a) and get the new index tree shown in Fig. 13. For a single broadcast channel, the broadcast is produced by the preorder traversal of the index tree. In the broadcast, the data nodes with the same parent will be allocated in adjacent positions in the broadcast. The time complexity for sorting the index tree is O(*Nlog m*), where *N* is the number of nodes in the index tree and *m* is the fanout.

A procedure to allocate the sorted index tree into any *k* broadcast channels is listed below. First, each node in the sorted index tree is associated with a number denoting the level of the node in the index tree. Then, we traverse the sorted index tree in preorder to produce a linear sequence. We give each node in the linear sequence a sequence number. Then, the linear sequence is stored in a data structure with an array of lists (*alcArray*), where nodes with the same level number are listed together in ascending order of their sequence numbers.

Procedure. 1_To_k_BroadcastChannel
{*alcArray[i].node indicates the first node of the list of level i. ptr→next indicates the next node of the node pointed by ptr in the list. C(i, j) denotes the jth slot of channel i.***}**

Begin

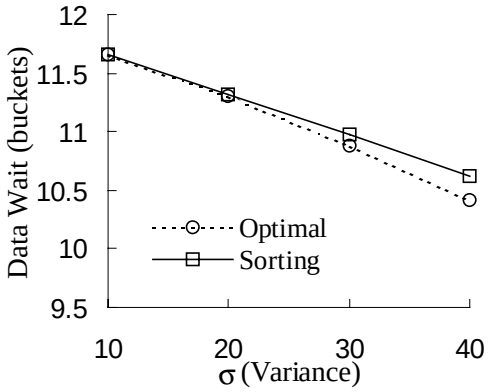
1: $i = j = 1$;
2: *level* =1;
3: *ptr* = *alcArray*[*level*].*node*;
4: $C(i, j) \leftarrow ptr$;
5: *level* = *level* + 1;
6: **While** (*level* <> DepthOfTree)
7: *ptr* = *alcArray*[*level*].*node*
8: $j = j + 1$; $C(i, j) \leftarrow ptr$;
9: *ptr* = *ptr*→*next*;
10: **While** (i <> NumOfChannels) AND (*ptr* <> Null)
11: $i = i + 1$; $C(i, j) \leftarrow ptr$;
12: *ptr* = *ptr*→*next*;
13: **End While**
14: *level* = *level* + 1;
15: **If** (*ptr* <> Null) **Then**
16: Merge(*alcArray*[*level*].*node*, *ptr*);
17: **End If**
18: $i = 1$;
19: **End While**
20: DumpList(*alcArray*[*level*].*node*);

End.

In the procedure, we scan each level of lists from the top and allocate the nodes in a list into the same slots of channels (statements 1~19). If the channel slots are not enough to hold all the nodes in a list, the remaining unallocated nodes in the list are merged into the list of the next level (statements 15~17). Finally, we dump all the nodes in the list of the last level into channel slots (statements 20). Assume the total number of nodes in the index tree be n. The main operations of the procedure are scanning and merging the lists which take totally $O(n)$ operations, respectively. Therefore, the transformation takes only linear time.

An experiment was performed to show the effect of the index tree sorting and the corresponding single channel allocation by using a full balanced m-nary tree with depth 3 as in Section 4.1. The access frequencies of data nodes are generated using the normal distribution $N(\mu, \sigma)$. In Fig. 14, we observe that when the fanout is small and the access frequencies are about the same, Sorting performs near Optimal. The reason is that in this situation the data nodes with the same parent prone to be

adjacent in the optimal broadcast allocation, which coincides with the preorder traversal in the sorted index tree.



$\mu = 100$ and $m = 4$

Fig. 14. The Performance of the Index Tree Sorting.

5. CONCLUSION

In this paper, we consider the optimization problem of the index and data allocation for any number of broadcast channels, given the constraint that no data replicates in a broadcast cycle. We show that the problem can be transformed to the Personnel Assignment Problem. By using similar techniques, the solution space of the allocation problem can be represented as a tree structure. We derive some properties to prune the nodes in the tree. A specific consideration on a single broadcast channel is also presented, where the tree space can be further reduced. Although the pruning technique can greatly reduce the tree space, the complexity of the solution is still high for a large index tree. Two heuristics are therefore proposed. The developed methodology in this paper can be applied to other scheduling problems.

Three more factors will be considered in the future. The first is to consider the change of access patterns on the broadcast data. If the change is frequent, an efficient on-line algorithm to immediately reflect the current broadcasting state is needed. The second is to allow data or index replication in a broadcast cycle. The access of broadcast data has to be initiated from the bucket containing the root of an index tree. To reduce the initial time after tuning to the broadcast channel, index nodes should be properly replicated and well organized. Finally, the third is to consider the allocation problem with an arbitrary graph representing the dependencies among broadcast data. For an index tree, there is a

hierarchical dependency. In [CHK99], the case for an acyclic directed graph is considered, where only one broadcast channel is used. A heuristic is proposed based on some allocation rules. We plan to develop an efficient algorithm for an arbitrary graph based on our proposed techniques.

REFERENCE

[Ach95] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data management for Asymmetric Communication Environments," *Proc. ACM SIGMOD Conf.*, pp. 199-210, San Jose, CA, May 1995.

[AFZ96] S. Acharya, M. Franklin, and S. Zdonik, "Disseminating Updated on Broadcast Disks," *Proc. 22nd VLDB Conf.*, pp. 354-365, Bombay, India, September 1996.

[CHK99] Y. C. Chehadeh, A. R. Hurson and M. Kavehrad, "Object Organization on a Single Broadcast Channel in the Mobile Computing Environment," *Multimedia Tools and Applications*, Vol. 9, No. 1, July 1999.

[CYW97] M. Chen, P. Yu, and K. Wu, "Indexed Sequential Data Broadcasting in Wireless Mobile Computing," *17th IEEE International Conference on Distributed Computing System*s, pp. 124-131, Baltimore, Maryland, May 1997.

[DCK97] A. Datta, A. Celik, J. Kim, and D. E. VanderMeer, "Adaptive Broadcast Protocols to Support Power Conservant Retrieval by Mobile Users," *13th International Conference on Data Engineering*, pp. 124-133, Birmingham, UK, April 1997.

[Fra98] M. Franklin, "Data in Your Face: Push Technology in Perspective," *Proc. ACM SIGMOD Conf.*, pp. 516-519, Seattle, Washington, June 1998.

[HT71] T. C. Hu and A. C. Tucker, "Optimal Computer Search Trees and Variable-length Alphabetic Codes," *SIAM J. Appl. Math.*, 21(4):514-532, 1971.

[IV94] T. Imielinski and S. Viswanathan, "Adaptive Wireless Information Systems," *Proc. Special Interest Group on Database Systems (SIGDBS) Conf.*, pp. 19-41, Tokyo, Japan, October 1994.

[IVB94a] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Data on Air: Organization and Access," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 9, No. 3, pp. 353-372, May/June 1997.

[IVB94b] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Energy Efficient Indexing on Air," *Proc. ACM SIGMOD Conf.*, pp. 25-36, Minneapolis, Minnesota, May 1994.

[IVB94c] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Power Efficient Filtering of Data on Air," *4th International Conference on Extending Database Technology (EDBT)*, pp. 245-258, Cambridge, England, March 1994.

[LC97] S. C. Lo and A.L.P. Chen, "An Adaptive Access Method for Broadcast Data", To appear in *IEEE Transactions on Knowledge and Data Engineering*, 1997.

[LL96] W. C. Lee and D. L. Lee, "Using Signature Techniques for Information Filtering in Wireless and Mobile Environments," *Distributed and Parallel Databases*, Vol. 4, No. 3, pp. 205-227, July 1996.

[LS95] H. V. Leong and A. Si, "Data Broadcasting Strategies over Multiple Unreliable Wireless Channels," *Proc. 4th International Conference on Information and Knowledge Management*, pp. 96-104, ACM, November 1995.

[Str89] J. K. Strayer, "Linear programming and its applications," *New York Springer-Verlag*, 1989.

[SRB97] K. Stathatos, N. Roussopoulos, and J. S. Baras, "Adaptive Data Broadcast in Hybrid Networks," *Proc. 23rd VLDB Conf.*, pp. 326-335, Athens, Greece, August 1997.

[SV96] N. Shivakumar and S. Venkatasubramanian, "Energy-Efficient Indexing For Information Dissemination In Wireless Systems," *ACM, Journal of Wireless and Nomadic Application*, 1996.

[TO98] K. L. Tan and B. C. Ooi, "On Selective Tuning in Unreliable Wireless Channels," *Data and Knowledge Engineering*, Vol. 28, No. 2, pp.209-231, 1998.

[TY96] K. L. Tan and J. X. Yu, "Energy Efficient Filtering of Nonuniform Broadcast," 1*6th IEEE International Conference on Distributed Computing Systems*, pp. 520-527, 1996.

[TY97] K. L. Tan and J. X. Yu, "An Analysis of Selective Tuning Schemes for Nonuniform Broadcast," *Data and Knowledge Engineering*, Vol. 22, No. 3, pp.319-344, 1997.

[TY98] K. L. Tan and J. X. Yu, "Generating Broadcast Programs that Support Range Queries," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 10, No. 4, pp. 668-672, 1998.

APPENDIX

Algorithm (Generating a reduced *k*-channel topological tree)

Input. An index tree with data items on the leaf nodes.

Output. A reduced *k*-channel topological tree.

Begin

1) Create the root of the topological tree, which contains the root of the index tree.

2) For each leaf node *P* of the topological tree

3) **Step 1**. (**Collect candidates as the elements of the next-neighbors**)

Collect set $S = \bigcup_{y \in PATH_T(P)} Children(y) - PATH_T(P)$.

4) **Step 2**. (**Prune the candidate set**)

5) Consider the following two cases to reduce the elements in *S*.

6) **Case 1**. The elements of *P* are all index nodes:

7) (i) If $k = 1$, then remove all the elements which are not the children of an element in *P* from *S*. Among the data nodes, leave only the one with the largest weight.

8) (ii) If $k \neq 1$, then remove all the data nodes that are not the children of any element in *P* from *S*. Among the remaining data nodes, leave the first *k*

large nodes in terms of their access frequencies.

9) **Case 2**. Other cases:

10) Remove the data nodes that are not the children of any element in *P* and their weights are larger than that of a data node in *P* from *S*.

11) **Step 3**. (**Generate the possible next-neighbors**)

12) Generate all the *k*-component subsets of *S* by the following rules.

13) (i) If there are *n* data nodes in a subset, then these data nodes must be the first *n* large ones in terms of the access frequencies.

14) (ii) If the elements of *P* are all index nodes and $k \neq 1$, then at least one child of an element in *P* should be included in the subset.

15) **Step 4**. (**Prune the next-neighbors**)

16) For each *k*-component subset, check whether a local swap can be performed between the subset and *P*.

17) (i) If there is a data node in the subset that can be locally swapped with an index node in *P*, then eliminate this subset.

18) (ii) Assume there is an index node *x* in *P* and an index node *y* in the subset. If *x* can be swapped with *y* and $W(y) > W(x)$, then eliminate this subset.

19) **Step 5**. (**Create a node for each next-neighbor**)

20) For each remaining *k*-component subset, create a corresponding node as a next-neighbor.

21) End For

22) If *S* is empty in each iteration of the above loop then stop, otherwise go to statement 2).

End.