Data Allocation on Wireless Broadcast Channels for Efficient Query Processing

Guanling Lee, Shou-Chih Lo, and Arbee L.P. Chen, *Senior Member*, *IEEE*

Abstract—Data broadcast is an excellent method for efficient data dissemination in the mobile computing environment. The application domain of data broadcast will be widely expanded in the near future, where the client is expected to perform complex queries or transactions on the broadcast data. To reduce the access latency for processing the complex query, it is beneficial to place the data accessed in a query close to each other on the broadcast channel. In this paper, we propose an efficient algorithm to determine the allocation of the data on the broadcast channel such that frequently co-accessed data are not only allocated close to each other, but also in a particular order which optimizes the performance of query processing. Our mechanism is based on the well-known problem named *optimal linear ordering*. Experiments are performed to justify the benefit of our approach.

Index Terms—Database broadcasting, query processing, access time, tuning time, broadcast program.

1 Introduction

Rapid advances in wireless communications and software/hardware technologies enable a client carrying a mobile device to access information without the restriction of time and location. Broadcast-based information systems provide dissemination of information with a cost independent of the number of clients, which compensates for the limited bandwidth in wireless communications. Moreover, the clients can retrieve the broadcast data by just tuning to the broadcast channel, which results in a certain degree of energy saving. Therefore, data broadcast has become an attractive solution for information dissemination. Database broadcast is first addressed in the Datacycle project [12], where the communication medium is high-speed optical fiber. The queries are processed by a hardware device which filters the data on the channel. The Datacycle architecture is improved in [27] by maintaining only the needed data on the broadcast channel. Several forms of data broadcast have been used in commercial products [2].

Assume the data on the broadcast channel are composed of *data objects* which may correspond to web pages or relation tuples. A client submits a query to retrieve data objects from the broadcast channel. The query may access one data object (called *simple query*) or more than one data object (called *complex query*). Many approaches have been proposed to schedule data objects for efficient processing of simple queries. In [17], a broadcast program where the data objects are broadcast in a periodic fashion is proposed. According to the access frequencies of the data objects, some frequently accessed data objects can be replicated in the broadcast program to reduce the access time. The methods to replicate data objects are presented in [1], [13], [26]. Moreover, in [14], [16], [18], [19], [23], index techniques are used to reduce the tuning time. For efficient processing of complex queries, to allocate the data objects accessed together on the broadcast channel can also improve the performance. As discussed and justified in [6], the traditional disk-based data allocation techniques perform poorly for the broadcast data due to the lack of the random-access feature on the broadcast channel. New channel-based data allocation techniques should be studied.

There exist relationships among the data objects to be broadcast. For example, the anchor relationship for the web pages, the referential integrity constraint for the relations in the relational database, and the composition relationship for the objects in the object database. In these cases, the related data objects for a complex query should be allocated in an order according to their relationships for a better performance, which complicates the data allocation problem. The issues of database broadcast in the mobile environment are studied in [24]. The data objects on the broadcast channel are relations in a relational database or classes in an object database. As mentioned before, clustering the data objects accessed in the complex queries frequently submitted can reduce the average access cost for processing the queries. The objective in [24] is to find an optimal broadcast order of the data objects such that the average access cost for a set of queries is minimized. This problem is formulated by a graph-based model. The optimal broadcast order is found by a branch-and-bound searching algorithm. However, as the number of data objects increases, the time needed to compute the optimal broadcast order increases exponentially. In fact, this kind of ordering problems can be proven to be NP-complete through the *optimal linear ordering* problem [8]. In [5], the method for finding the optimal broadcast program for two dependent files is proposed. In [4], a lower bound on the average access time of the optimal

- *G. Lee is with the Department of Computer Science and Information Engineering, National Dong Hwa University, 1, Sec. 2, Da Hsueh Rd., Shou-Feng, Hualien, Taiwan 973, Republic of China. E-mail: guanling@mail.ndhu.edu.tw.*
- *S.-C. Lo and A.L.P. Chen are with the Department of Computer Science, National Tsing Hua University, 101, Section 2 Kuang Fu Road, Hsinchu, Taiwan 300, Republic of China. E-mail: {robert, alpchen}@cs.nthu.edu.tw.*

Manuscript received 15 July 2001; revised 15 May 2002; accepted 20 May 2002.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 116591.

broadcast program for the complex queries is derived. Moreover, an algorithm to achieve a random permutation of the broadcast data is proposed whose corresponding average access time is twice of the lower bound on the average access time. A special case where there is no cyclic dependence among the dependent data is discussed in [6]. The broadcast order is decided by a set of heuristic rules. In [7], the scheduling method for answering complex queries where there is no access order constraint among the required data objects is presented. The broadcast order is decided by a greedy method based on the frequencies of queries. Based on [7], [21], [22] propose a more efficient algorithm to solve this problem. The index issues for answering complex queries are discussed in [9], where the client always waits for the index placed at the beginning of the broadcast cycle before any data access. In [11], the issue of allocating dependent data on multiple channels is discussed. A heuristic algorithm is proposed to cluster related data objects to minimize overall broadcast time.

In this paper, database broadcasting with query optimization is considered. To measure the cost of query processing, two metrics introduced in [16] can be used. The *access time* is the time elapsed from the moment a client first tunes in the broadcast channel to the moment all the relevant data are downloaded. The *tuning time* is the time spent by the client listening to the broadcast channel, which is an indicator of the power consumption. To reduce the access time, relevant attributes accessed in a query should be allocated nearby in the broadcast channel. To reduce the tuning time, the amount of data involved in the query processing should be small. In our approach, a relational database is first vertically partitioned into fragments based on attributes. Given the information of a set of complex queries with their querying frequencies in the past, we predict future data accesses and allocate relevant attributes instead of the whole database on the broadcast channel. A client can retrieve the attributes involved in the query by directly listening to the broadcast channel. The query processing is performed during the access of the relevant attributes. For the case where the needed attribute is not allocated on the broadcast channel, the client can submit the query to the server and receive the needed data on the on-demand channel. Our problem is to allocate the attributes on the broadcast channel such that the average access time to access the attributes involved in the queries according to the query optimization order is minimized. Accessing attributes and processing queries according to the query optimization order minimizes the amount of downloaded data and, therefore, minimizes the tuning time.

The rest of the paper is organized as follows: Section 2 discusses the database broadcasting issues and introduces some existing problems related to our approach. A graph representation method for solving our problem is discussed in Section 3. In Section 4, how to allocate attributes on the broadcast channel is presented. A simulation model and the analysis of the simulation results are described in Section 5. Finally, Section 6 concludes this work.

2 Preliminary Concepts

2.1 Issues on Database Broadcasting

In our approach, a relational database is first vertically partitioned into fragments based on attributes. The values of each attribute in a relation are sequentially allocated in the broadcast channel. To integrate the attribute values of a tuple from the broadcast channel, each attribute value is associated with a tuple number indicating the tuple the attribute value belongs to. As mentioned above, the clients access the attributes involved in the queries according to the query optimization order. In conventional query optimization, the selection operations are performed first to reduce the size of the temporary results. Therefore, for reducing the tuning time, the values of the *select attributes* should be retrieved to process prior to the values of the other attributes in the broadcast channel.

We transform the query in SQL into a *query pattern* in the format [SA, JA, PA], where SA (select attributes) is the set of attributes in the where-clause by the format x θ c, where x is an attribute, θ is a comparison operator, and c is a constant, JA (*join attributes*) is the set of attributes in the where-clause by the format x θ y, where x and y are attributes, and PA (*project attributes*) is the set of attributes in the select-clause. If the intersection of SA, JA, and PA is not empty, we remove the duplicate attributes from the sets in the order of PA, JA, and SA. The square bracket of the query pattern indicates the query processing order. That is, attributes in SA should be accessed before attributes in JA and attributes in PA should be accessed last. Among the attributes in SA, JA, or PA, there is no order constraint. For example, assume there are three relations to be broadcast. The corresponding attributes of each relation are listed below.

$$\text{relation A} = (a_1, a_2), \text{relation B} = (b_1, b_2),$$
$$\text{relation C} = (c_1, c_2).$$

A broadcast order of the attributes is called a *broadcast program*. $a_1b_1b_2a_2c_1c_2$ or $a_1b_1a_2c_2c_1b_2$ are both possible broadcast programs. Consider the following query:

Select a_2, b_2
From A, B
Where $a_1 < 20$ and $a_1 = b_1$

In this query, $SA = \{a_1\}$, $JA = \{b_1\}$ (because $a_1 \in SA$, it is removed from JA), and $PA = \{a_2, b_2\}$. The client first tunes in the broadcast channel to download a_1 and perform the selection operation on the values of attribute a_1. Then, the client downloads b_1 and performs the join operation with the values of the selection results of attribute a_1. After that, we get pairs of tuple numbers denoting the tuples in relations A and B which are joined together and satisfy the query conditions in the where-clause. Through these tuple numbers, the relevant values of attributes a_2 and b_2 are then downloaded to be the answer of the query.

2.2 Problem Formulation

In our approach, an *access graph* is used to represent the order among the attributes. Our data allocation algorithm will be developed based on the access graph. An access graph is a directed weighted graph, where each node of the access graph represents an attribute and each edge e_{ij} is

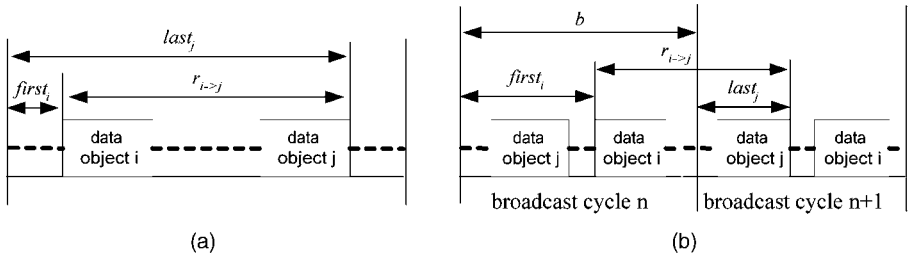

Fig. 1. The offset from data object i to data object j. (a) Data object i appears before data object j. (b) Data object i appears after data object j.

associated with a weight which denotes the total frequencies of the accesses from attribute i to attribute j. Notice that cycles can exist in the access graph. Given a set of query patterns, in Section 3, the method of transforming a set of query patterns into an access graph will be presented. In the following, the concepts used in our approach are introduced. Moreover, for easier presentation, attributes appearing in SA, JA, or PA are all called data objects.

Given an access graph G(V, E), where V is the set of vertices and E is the set of edges. Let $\mathrm{w}(\mathrm{e_{ij}})$ be the weight of edge $\mathrm{e_{ij}}$ (the edge directed from vertex i to vertex j). Moreover, let $\|\mathrm{i}\|$ be the size (number of data buckets) of data object i associated with vertex i. Assume a client tunes in the channel at random during a broadcast cycle. If the client tunes in the channel after the start of the broadcast of the requested data objects, the client has to wait for the broadcast of the data objects in the next broadcast cycle. Our goal is to find an optimal broadcast order of the data objects in the access graph. The optimal broadcast order is the order with minimum average access time

$$(1/b) \times \sum_{k=0}^{b-1} \sum_{e_{ij} \in E} \left(\left(w(e_{ij}) / \sum_{e_{ij} \in E} w(e_{ij}) \right) \times (k + r_{i \to j}) \right),$$

where b denotes the total number of data buckets, i.e.,

$$\sum_{x \in V} \|x\|, w(e_{ij}) / \sum_{e_{ij} \in E} w(e_{ij})$$

is used to normalize the weights in the access graph, k denotes the offset from the bucket the client first tunes in the channel to the first bucket of data object i, and $r_{i \to j}$ denotes the offset from the first bucket of data object i to the last bucket of data object j. Notice that, since data object i can be allocated before or after data object j in the broadcast channel, the offset from data object i to data object j can be computed in two ways, as shown in Fig. 1.

For the case where data object i appears before data object j, $\mathrm{r_{i \to j}}$ can be computed by $(last_j - first_i)$, while, for the case where data object i appears after data object j, $\mathrm{r_{i \to j}}$ can be computed by $(b - first_i + last_j)$, where $last_j$ is the offset from the first bucket of the broadcast channel to the last bucket of data object j and $first_i$ is the offset from the first bucket of the broadcast channel to the first bucket of data object i. We define the *optimal cycle ordering* problem as follows:

Optimal cycle ordering problem: Given an access graph G(V, E), the problem is to find a one-to-one function f: $\mathrm{V} \to \{1, 2, 3, \ldots, |\mathrm{V}|\}$ such that $\sum_{e_{ij} \in E} w(e_{ij}) \times r_{i \to j}$ (denoted as costcycle) is minimized, where

$$r_{i \to j} = (last_j - first_i + b) \bmod b,$$
$$last_j = \sum_{x \in V and f(x) \le f(j)} \|x\|,$$

and

$$first_i = \sum_{x \in V and f(x) < f(i)} \|x\|.$$

Lemma 1. *For an access graph, its corresponding optimal cycle order (OCO) is the optimal data broadcast order.*

Proof. Refer to [20]. □

There exists a problem named *optimal linear ordering*, which is similar to the optimal cycle ordering problem. In the following, the definition of the optimal linear ordering problem and the relationship between these two problems are presented.

Optimal linear ordering problem [3]: Given a weighted directed graph G(V, E), where V is the set of vertices and E is the set of edges. Let $\mathrm{w}(\mathrm{e_{ij}})$ be the weight of edge $\mathrm{e_{ij}}$ (the edge directed from vertex i to vertex j). The optimal linear ordering problem is to find a one-to-one function f: $\mathrm{V} \to \{1, 2, 3, \ldots, |\mathrm{V}|\}$ such that $f(i) < f(j)$ whenever $\mathrm{e_{ij}} \in \mathrm{E}$ and such that $\sum_{e_{ij} \in E} w(e_{ij}) \times h_{i \to j}$ is minimized, where $h_{i \to j} = f(j) - f(i)$.

The problem is NP-complete, but is solvable in polynomial time if G is a tree. The detailed algorithm to determine the *optimal linear order* (OLO) of a tree can be found in [3]. In the following, an important property in [3] is presented.

Property 1. *Let ρ^* be an OLO for a tree T. If T' is a tree with a subtree identical to T or T' is formed by adding new children to the root of T, then there exists an OLO ρ' for T' in which the relative order of ρ^* is preserved.*

The original optimal linear ordering problem takes the vertices with equal size. To deal with the vertices with different sizes, we only need to change the function $\mathrm{h_{i \to j}}$ to

$$\left(\sum_{x \in V and f(x) \le f(j)} \|x\| - \sum_{x \in V and f(x) \le f(i)} \|x\| \right),$$

where $\|\mathrm{x}\|$ denotes the size of vertex x. The meaning of the new $\mathrm{h_{i \to j}}$ is shown in Fig. 2. With a slight modification, the

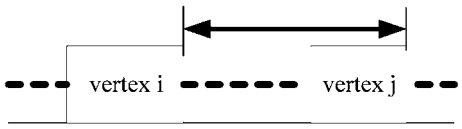

Fig. 2. The meaning of the new $h_{i\to j}$.

algorithm proposed in [3] can be used to deal with the vertices with different sizes. We do not further discuss the size issue in the following.

Because of the cyclic property of the optimal cycle ordering problem, the constraint "$f(i) < f(j)$ whenever $e_{ij} \in E$" does not exist in the optimal cycle ordering problem. However, if the property "$f(i) < f(j)$ whenever $e_{ij} \in E$" is held in the OCO of an access graph, the OCO of the access graph is the same as the OLO of the access graph.

Lemma 2. *In the optimal cycle ordering problem, if $f(i) < f(j)$ for each $e_{ij} \in E$ can be guaranteed in the given access graph, then the OCO of the graph is the same as the OLO of the graph.*

Proof. Refer to [20]. □

For an arbitrary access graph, the property of "$f(i) < f(j)$ for each $e_{ij} \in E$" does not always hold. However, if the graph is a tree, it must be true. Therefore, we can transform the access graph to a forest (named *access forest*) by removing some edges, apply the optimal linear ordering algorithm on the access forest, then consider the removed edges to approach the optimal data broadcast problem.

To transform an access graph to an access forest, which keeps as much information as possible, an algorithm named *maximum branching* can be used.

Maximum branching problem [25]: Consider a weighted directed graph G (V, E), where V is the set of vertices and E is the set of edges. Let $w(e_{ij})$ be the weight of edge e_{ij} (the edge directed from vertex i to vertex j) and W(G) be the sum of the weights of all the edges in G. A subgraph G_b containing all vertices of G is a *branching* of G if G_b has no directed cycles and the in-degree of each vertex in G_b is at most 1. Clearly, each connected component of G_b is a tree and G_b is a forest. The G_b with a maximum $W(G_b)$ is called a *maximum branching*. The detailed algorithm to find the maximum branching can be found in [25]. In the following, an important property which will be used to transform a set of query patterns to an access graph is presented.

Property 2. *Let $\{e_{ix}\}$ be the set of edges which point to vertex x. Among the edges in $\{e_{ix}\}$, which are not contained in a cycle, the edge with the maximum weight will be selected to be in the maximum branching.*

3 Representing Query Patterns as an Access Graph

In this section, how to represent a set of query patterns as an access graph is discussed. A query pattern contains an ordered access triple [SA, JA, PA]. Notice that there is no access order constraint among the data objects in SA, JA, or PA. As mentioned above, an access graph is a directed weighted graph, where each node represents a data object and each edge e_{ij} represents the access order from data object i to data object j. The weight associated with e_{ij} denotes the total frequencies of the accesses from data object i to data object j. Therefore, to transform a set of query patterns to an access graph, the access order of the data objects in SA, JA, and PA should be determined.

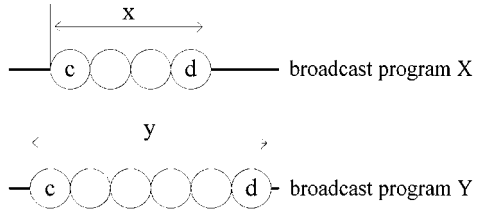

Fig. 3. Two broadcast programs.

There are two steps to determine the access order. The first step makes use of the known access order of some data object pairs and the second step makes use of Property 2 given in Section 2.

3.1 Step 1 of the Access Order Determination Process

The following lemma will be used when determining the access order:

Lemma 3. *Given a set of ordered access pairs, assume there exist two pairs, say $[c, d]$ and $[d, c]$ with access frequencies f_{cd} and f_{dc}, respectively. If $f_{cd} > f_{dc}$, then the two pairs can be replaced by the ordered access pair $[c, d]$ (named a replacement pair) with access frequency $f_{cd} - f_{dc}$ without affecting the derivation of an optimal broadcast program.*

Proof. Fig. 3 shows two broadcast programs X and Y containing data objects c and d. For the original set of ordered access pairs, the difference of the average access times for broadcast program X and broadcast program Y is

$$\begin{aligned}
&(1/b) \times (1/W) \times \left(\sum_{i=0}^{b-1} (f_{cd} \times (i + x) \right) \\
&+ \sum_{i=0}^{b-1} (f_{dc} \times (i + b - x + ||c|| + ||d||))) \\
&- (1/b) \times (1/W) \times \left(\sum_{i=0}^{b-1} (f_{cd} \times (i + y) \right) \\
&+ \sum_{i=0}^{b-1} (f_{dc} \times (i + b - y + ||c|| + ||d||))) \\
&= (1/W) \times ((f_{cd} - f_{dc}) \times x - (f_{cd} - f_{dc}) \times y),
\end{aligned}$$

where x denotes $r_{c\to d}$ in broadcast program X, y denotes $r_{c\to d}$ in broadcast program Y, and W denotes the summation of the access frequencies of the ordered access pairs in the original set of ordered access pairs.

For the ordered access pairs containing the replacement pair, the difference of the average access times for broadcast program X and broadcast program Y is

$$(1/b) \times (1/W') \times \sum_{i=0}^{b-1} (f_{cd} - f_{dc}) \times (i + x)$$
$$- (1/b) \times (1/W') \times \sum_{i=0}^{b-1} (f_{cd} - f_{dc}) \times (i + y)$$
$$= (1/W') \times ((f_{cd} - f_{dc}) \times x - (f_{cd} - f_{dc}) \times y),$$

where W′ denotes the summation of the access frequencies of the ordered access pairs containing the replacement pair.

Because both W and W′ are positive numbers, the sign of the two differences is the same. Therefore, the optimal broadcast programs for the original set of ordered access pairs and the one containing the replacement pair are the same. □

We use an example to illustrate how Lemma 3 works.

Example 1. Given query $\text{pattern}_1 = [\{a, f\}, \{b, c\}, \{d, e\}]$ with access frequency $f_1 = 20$, query $\text{pattern}_2 = [\{c\}, \{a, d\}, \{b, e\}]$ with access frequency $f_2 = 30$, and query $\text{pattern}_3 = [\{e\}, \{d\}, \{h\}]$ with access frequency $f_3 = 10$. In Step 1 of the access order determination process, the known access order is used to determine the access order of the data objects in {}. According to query pattern_1, we know data object b should be accessed before data object e. Therefore, query pattern_2 can be revised to $[\{c\}, \{a, d\}, [b, e]]$. According to query pattern_2, we know data object d should be accessed before data object e. However, according to query pattern_3, data object e should be accessed before data object d. By Lemma 3, we know that data object d should be accessed before data object e with an access frequency 30-10. Therefore, query pattern1 can be revised to $[\{a, f\}, \{b, c\}, [d, e]]$.

3.2 Step 2 of the Access Order Determination Process

In this step, the revised query patterns are decomposed into a set of ordered access pairs to construct a *temporary access graph*. The decomposition process identifies each ordered access pair from the set of revised query patterns. Moreover, the temporary access graph will be used to determine the access order of the data objects in the remaining {}. The decomposition process is illustrated as follows:

Query Pattern Decomposition
(input: the set of revised query patterns, output: a set of ordered access pairs (OAP))

1. Let OAP = {}
2. For each QP in the set of revised query patterns
   For each S->D in QP /* S->D is SA->JA or JA->PA*/
   If S is an unordered set
   If D is an unordered set
   For each data object a in S
   For each data object b in D
   OAP = OAP ∪ {[a, b]}
   Else
   For each data object a in S
   OAP = OAP ∪ {[a, b]} where b is the first data object in D
   OAP = OAP ∪ {D}
   Else
   Let a = the last data object in S
   OAP = OAP ∪ {S}
   If D is an unordered set
   For each data object b in D
   OAP = OAP ∪ {[a, b]}
   Else
   OAP = OAP ∪ {[a, b]} where b is the first data object in D
   OAP = OAP ∪ {D}
3. Output OAP

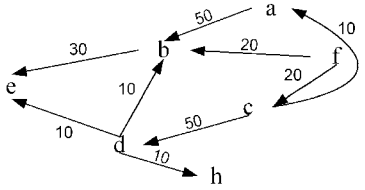

Fig. 4. The temporary access graph constructed from the three query patterns.

Continuing the above example, query pattern_1 is decomposed to {[a, b], [a, c], [f, b], [f, c], [b, d], [c, d], [d, e]}, each with access frequency 20, query pattern_2 is decomposed to {[c, a], [c, d], [a, b], [d, b], [b, e]}, each with access frequency 30, and query pattern_3 is decomposed to {[e, d], [d, h]}, each with access frequency 10. Merging the decomposition results, we get [a, b], [c, d], each with access frequency 20+30, [c, a], [d, b], each with access frequency 30-20, [d, e] with access frequency 20-10, [f, b], [f, c], each with access frequency 20, [b, e] with access frequency 30, and [d, h] with access frequency 10. According to the merging result, a temporary access graph can be constructed. Fig. 4 shows the temporary access graph constructed from the three query patterns.

The temporary access graph will be used to determine the access order of the data objects in the remaining {} to construct the access graph. In the following, an important property for determining the access order is presented. As mentioned above, the maximum branching algorithm will be used to transform the access graph into an access forest. To produce a better access forest, we should consider the property of the maximum branching algorithm when constructing the access graph. Given a temporary access graph G(V, E), according to Property 2, we define *MIW* (Maximum In-edge Weight) for each $x \in V$ as follows:

$$MIW(x) = \begin{cases} \text{the maximum weight among those weights associated with} \\ \text{the set of } [y,\ x] \text{ where } [y,\ x] \text{ is not contained in any circuit.} \\ \\ 0, \text{ if there is no } [y,\ x] \text{ satisfying the above condition.} \end{cases}$$

The following lemma is used to illustrate how MIW works to determine the access order of the data objects in {} of the revised query patterns.

Lemma 4. *Given a temporary access graph $G(V, E)$ where $x, y \in V$, $e_{xy}, e_{yx} \notin E$, and $MIW(x) > MIW(y)$. Assume*

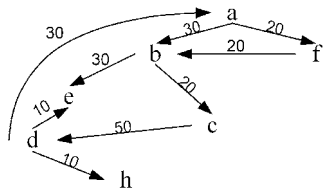

Fig. 5. Access graph constructed from the tree query patterns.

e_{xy} or e_{yx} with access frequency f can be added into G. If e_{yx} is added into G, which can be preserved after applying the maximum branching algorithm, then, by adding e_{xy} instead of e_{yx} into G, e_{xy} can also be preserved.

Proof. Refer to [20]. □

Given {x, y} in a revised query pattern, if MIW(x) > MIW(y), then {x, y} will be turned into [x, y]. Continuing the above example, the access order of the data objects in {a, f}, {b, c}, and {a, d} needs to be determined. To determine the access order, we first compute the MIW of the above data objects. Referring to Fig. 4, we get MIW(a) = 10, MIW(f) = 0, MIW(b) = 50, MIW(c) = 20, and MIW(d) = 50. By Lemma 4, we get [a, f], [b, c], and [d, a], and query $\mathrm{pattern}_1 = [[\mathrm{a}, \mathrm{f}], [\mathrm{b}, \mathrm{c}], [\mathrm{d}, \mathrm{e}]]$, query $\mathrm{pattern}_2 = [\mathrm{c}, [\mathrm{d}, \mathrm{a}], [\mathrm{b}, \mathrm{e}]]$, and query $\mathrm{pattern}_3 = [\mathrm{e}, \mathrm{d}, \mathrm{h}]$. Fig. 5 shows the access graph constructed for the three query patterns.

The process of transforming the set of query patterns to an access graph is summarized as follows:

1. Use known access orders to determine the order of the data objects in SA, JA, and PA to revise the query patterns.
2. Decompose the revised query patterns into a set of ordered access pairs.
3. Construct the temporary access graph from the set of ordered access pairs.
4. Compute the MIW to determine the access order of the data objects whose access order is not yet determined.
5. Construct the access graph.

4 The Scheduling Algorithm

In our approach, the maximum branching algorithm is used to transform an access graph to an access forest. The time complexity of the maximum branching algorithm is $O(|E|log|V|)$ [25]. Therefore, the transformation process can be done in polynomial time. An example is shown in Fig. 6. In the example, the sizes of the vertices in the access graph are all set to 1.

After the access forest is produced, we determine the OLO of each tree (named *access tree*) in the access forest and concatenate the OLOs to form the result. However, the information loss induced by the removed edges in the transformation process has to be considered to get the final broadcast order. There are three cases to consider. For the first case, the information loss can be avoided by refining the access graph. The details will be discussed in Section 4.1. In the second case, the starting and ending vertices of the removed edges are in the same access tree (named *intraedge*) such as a->d in Fig. 6. We consider how to reorder the vertices to get a smaller average access time. The reordering method will be discussed in Section 4.2. For the third case, the starting and ending vertices of the removed edges are in different access trees (named *interedge*) such as m->i. We consider how to merge the OLOs of the access trees to get a smaller average access time. The merging method will be discussed in Section 4.3. The flow of our approach is shown in Fig. 7.

4.1 Refining Access Graph

If the number of the edges removed by the maximum branching algorithm can be reduced, the information kept in the access forest can be increased. The goal of refining the access graph is to modify the access graph such that the refined access graph keeps the same information as the original access graph but the number of edges to be removed by the maximum branching algorithm is reduced.

The access graph shown in Fig. 8 is called a Δ *graph*. The definition of a Δ graph is as follows:

Definition 1. *A Δ graph is an acyclic directed weighted graph* $(\mathrm{V}_\Delta, \mathrm{E}_\Delta)$, *where* $\mathrm{V}_\Delta = \{\mathrm{r}, \mathrm{m}, \mathrm{e}\}$ *and* $\mathrm{E}_\Delta = \{\mathrm{e_{rm}}, \mathrm{e_{re}}, \mathrm{e_{me}}\}$. r *is called the* root node *of the Δ graph.*

The following lemma shows that, for a Δ graph, the OCO can be determined by a simple statement.

Lemma 5. *Given a Δ graph, the OCO is "rme" if* $\|e\| \times w(e_{rm}) + \|r\| \times w(e_{me}) \geq \|m\| \times w(e_{re})$; *otherwise, the OCO is "rem."*

Proof. Refer to [20]. □

According to Lemma 5, we know which edge can be removed without affecting the optimal order. For example, in Fig. 8, if $\|\mathrm{r}\| = \|\mathrm{m}\| = \|\mathrm{e}\|$, the edge $\mathrm{e_{re}}$ should be removed and the access graph becomes an access tree. Applying the optimal linear ordering algorithm, the OCO "rme" will be

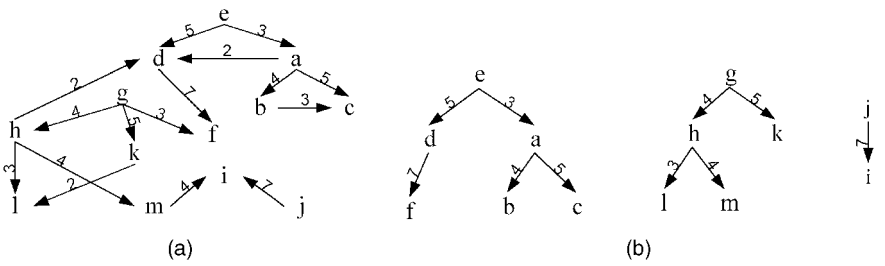

Fig. 6. (a) An access graph and (b) the access forest of (a).

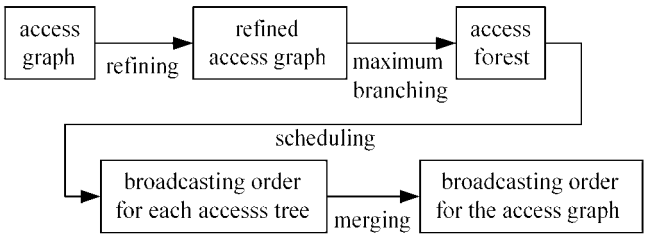

Fig. 7. The flow of our approach.

obtained. Refer to Property 1, adding new children or a new parent to node r does not affect the optimal order of the three nodes. Therefore, Lemma 5 can be extended to Lemma 6 which can deal with a more complex graph named Δ' *graph*.

Definition 2. *A Δ' graph is a directed weighted graph $(\mathrm{V}_{\Delta'}, \mathrm{E}_{\Delta'})$, which has at least one Δ graph as its subgraph and the root node of the Δ graph is a* cut node.

Lemma 6. *Given a Δ' graph, the OCO of the three nodes r, m, and e in the Δ graph is equal to the OLO of the three nodes in the graph modified as follows: If*

$$\|e\| \times w(e_{rm}) + \|r\| \times w(e_{me}) \geq \|m\| \times w(e_{re}),$$

then add $w(e_{re})$ to $w(e_{rm})$ and to $w(e_{me})$ and remove edge e_{re}. Otherwise, subtract $w(e_{me})$ from $w(e_{rm})$, add $w(e_{me})$ to $w(e_{re})$, and remove edge e_{me}. If $w(e_{rm}) < 0$, then remove edge e_{rm}, insert edge e_{mr}, and set $w(e_{mr})$ to $|w(e_{rm})|$.

Proof. Refer to [20]. □

For a Δ' graph (Δ graph is a special case of Δ' graph), we can modify the access graph according to Lemma 6 to get a refined access graph which can avoid information loss after executing the maximum branching algorithm. An example is shown in Fig. 9. Referring to Fig. 9a, $\|e\| \times w(e_{rm}) + \|r\| \times w(e_{me}) \geq \|m\| \times w(e_{re})$ $(4 \times 3 + 3 \times 2 > 2 \times 7)$, therefore, in the refined access graph, e_{re} is removed and $w(e_{rm})$ and $w(e_{me})$ are set to 3 + 7 and 2 + 7, respectively. For the case shown in Fig. 9b, $\|e\| \times w(e_{rm}) + \|r\| \times w(e_{me}) < \|m\| \times w(e_{re})$ $(1 \times 3 + 3 \times 2 < 2 \times 7)$, therefore, in the refined access graph, e_{me} is removed and $w(e_{rm})$ and $w(e_{re})$ are set to $3 - 2$ and $7 + 2$, respectively.

The refinement algorithm is presented as follows:

Access Graph Refinement Algorithm

1. For each subgraph $\mathrm{G_s}$ of the given access graph (V, E)

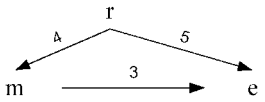

Fig. 8. A Δ graph.

2. If $\mathrm{G_s}$ is a Δ' graph
3. If $\|e\| \times w(e_{rm}) + \|r\| \times w(e_{me}) \geq \|m\| \times w(e_{re})$ then
4. $w(e_{rm}) = w(e_{rm}) + w(e_{re})$,
   $w(e_{me}) = w(e_{me}) + w(e_{re})$, $\mathrm{E} = \mathrm{E} - \{\mathrm{e_{re}}\}$.
5. Else
6. $w(e_{rm}) = w(e_{rm}) - w(e_{me})$,
   $w(e_{re}) = w(e_{re}) + w(e_{me})$, $\mathrm{E} = \mathrm{E} - \{\mathrm{e_{me}}\}$
7. If $w(e_{rm}) < 0$
8. $\mathrm{E} = \mathrm{E} - \{\mathrm{e_{rm}}\} \cup \{\mathrm{e_{mr}}\}$, $w(e_{mr}) = |w(e_{rm})|$.

In our approach, the Access Graph Refinement Algorithm is applied to the access graph first to get the refined access graph. Then, take the refined access graph as the input of the maximum branching algorithm to get an access forest. Fig. 10 shows the refined access graph and its maximum branching of the graph shown in Fig. 6. For more complex cases, the information loss cannot be avoided by simply modifying the access graph. Therefore, we record the edges removed when applying the maximum branching algorithm. As mentioned above, there are two kinds of removed edges (intraedge and interedge). We store the intraedges and interedges in $\mathrm{RE_{intra}}$ and $\mathrm{RE_{inter}}$, respectively. $\mathrm{RE_{intra}}$ and $\mathrm{RE_{inter}}$ will be further used to reduce the information loss.

4.2 Scheduling Access Tree

Our scheduling algorithm is based on the optimal linear ordering algorithm with a consideration of the edges in $\mathrm{RE_{intra}}$. We use a step-by-step method to solve the scheduling problem by considering each removed edge in $\mathrm{RE_{intra}}$.

Referring to Fig. 11, if we apply the optimal linear ordering algorithm on the access tree $(\mathrm{V_t}, \mathrm{E_t})$, the order of nodes a, b, and c will be determined. According to the order given by the optimal linear ordering algorithm and the direction indicated by the removed edge between node c and node b, two cases should be considered.

Case I: The order of the nodes indicated by the removed edge is the same as the order given by the optimal linear ordering algorithm.

For example, if the OLO of the access tree shown in Fig. 11 is $\mathrm{a} \ldots \mathrm{c} \ldots \mathrm{b}$ or $\mathrm{c} \ldots \mathrm{a} \ldots \mathrm{b}$, then the order of c, b is the

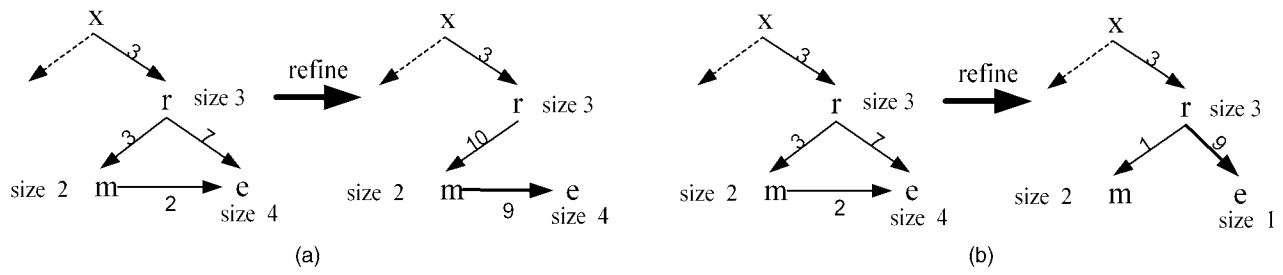

Fig. 9. Examples for Lemma 6.

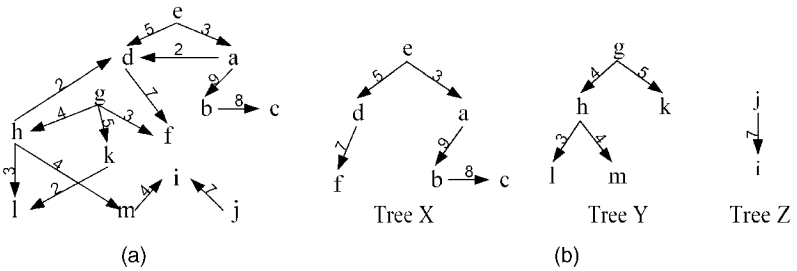

Fig. 10. (a) The refined access graph of the graph in Fig. 6 and (b) the access forest of (a).

same as the removed edge e_{cb}. This case can be further divided into two subcases:

Case I.a: According to the OLO of the access tree, the starting node (c) of the removed edge is between the ending node (b) and its parent node (a). For example, if the OLO of the access tree is $a \ldots c \ldots b$, then it is in Case I.a.

In this case, if there is no node between c and b, then we do nothing. The reason is that the removed edge is e_{cb} and, no matter what effort we make, we cannot make c and b get closer. If there exist some nodes between c and b, the access tree will be modified as follows: $E_t = E_t - \{e_{ab}\} \cup \{e_{cb}\}$, $w(e_{cb}) = w(e_{cb}) + w(e_{ab})$. According to the original access tree (before considering the removed edge), we know b appears after c. Therefore, if we make b closer to c (considering the removed edge), the average access time may be reduced. Therefore, we reconnect the removed edge e_{cb}, remove the edge e_{ab}, and add the weight of e_{ab} to the weight of e_{cb}. Notice that the access time will be underestimated because $r_{a\to b} > r_{c\to b}$ and $r_{c\to b}$ is used to approximate $r_{a\to b}$. After modifying the access tree, the optimal linear ordering algorithm is applied on the subtree rooted at node c to reschedule the subtree. If the average access time of the current broadcast program is smaller than that of the previous broadcast program, we use the modified access tree as the access tree to be further considered; otherwise, we use the previous access tree as the access tree for further consideration.

Case I.b: According to the OLO of the access tree, the starting node of the removed edge (c) appears before the parent node (a) of the ending node (b). For example, if the OLO of the access tree is $c \ldots a \ldots b$, then it is in Case I.b.

In this case, if there is no node between a and b, then we do nothing. The reason is that if the removed edge is e_{cb}, we cannot make c and b get closer without changing the order of a and b. If there exist some nodes between a and b, the access tree will be updated as follows: $w(e_{ab}) = w(e_{cb}) + w(e_{ab})$. According to the original access tree before considering the removed edge, we know a and b both appear after c. Therefore, if we make b closer to c without changing the order of a and b (i.e., making b closer to a), the average access time may be reduced. Therefore, we add the weight of e_{cb} to the weight of e_{ab}. Notice that the access time will be under estimated (because $r_{c\to b} > r_{a\to b}$, and $r_{a\to b}$ is used to approximate $r_{c\to b}$). After updating the access tree, the optimal linear ordering algorithm is applied on the subtree rooted at node a to reschedule the subtree. If the average access time of the current broadcast program is smaller than that of the previous broadcast program, we use the modified access tree as the access tree to be further considered; otherwise, we use the previous access tree as the access tree for further consideration. For example (refer to Fig. 10b), the OLO of tree Y is "gkhml," where the order of node k and node l is the same as the removed edge e_{kl}. It is in Case I.b. Moreover, there are nodes between node h and node l in the OLO. Tree Y is modified by updating $w(e_{hl})$ to 3 + 2. We apply the optimal linear ordering algorithm on the subtree rooted at node h. Fig. 12 shows the modified trees and the corresponding broadcast programs for the trees in Fig. 10b.

Case II: The order given by the optimal linear ordering algorithm is different from the direction indicated by the removed edge. For example, if the OLO of the access tree is $a \ldots b \ldots c$, then the order of b, c is different from the removed edge e_{cb}, it is in case II.

In this case, if there is no node between a and b, then we do nothing. The reason is that if the removed edge is e_{cb}, we cannot reduce $r_{c\to b}$ without changing the order of a and b. If there exist some nodes between a and b, the access tree will be updated as follows: $w(e_{ab}) = w(e_{cb}) + w(e_{ab})$. According to the original access tree (before considering the removed edge), we know a and b both appear before c. Therefore, if we reduce $r_{c\to b}$ without changing the order of a and b (i.e., making b closer to a), the average access time may be reduced. Therefore, we add the weight of e_{cb} to the weight of e_{ab}. After updating the access tree, the optimal linear ordering algorithm is applied on the subtree rooted at node a to reschedule theo subtree. If the average access time of the current broadcast program is smaller than that of the

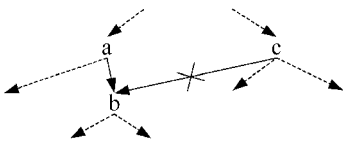

Fig. 11. A removed edge and its associated access tree.

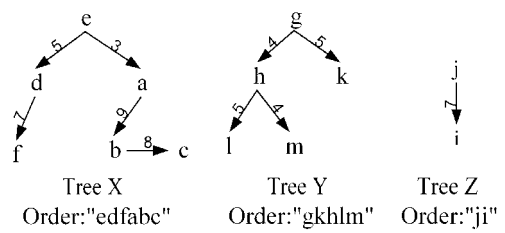

Fig. 12. The modified trees and the corresponding broadcast programs for the trees in Fig. 10b.

previous broadcast program, we use the modified access tree as the access tree to be further considered; otherwise, we use the previous access tree as the access tree for further consideration. Referring to Fig. 10b, the OLO of tree X is "edfabc," where the order of node d and node a is different from the removed edge e_{ad}. It is in Case II. However, there is no node between node e and node d in the OLO; therefore, we do nothing.

As mentioned above, we use a step-by-step method. Therefore, the execution order of the removed edges will affect the result of the broadcast program. We sort the removed edges in a decreasing order according to their weights to guarantee that the edges with larger weights will be considered first.

The scheduling algorithm is as follows:

Scheduling Algorithm

1. Apply the optimal linear ordering algorithm to the given access tree (V, E) and the output broadcast program is stored in *list*
2. Previous_average_access_time= average access time of the output broadcast program
3. While $\mathrm{RE}_{\mathrm{intra}}$ not empty
4. Remove the edge with the largest weight, say $\mathrm{e_{cb}}$, from $\mathrm{RE}_{\mathrm{intra}}$
5. Consider the order of c, b and b's parent node, say a, in the *list*
6. If it falls in Case I.a and there exist some nodes between c and b in the *list*
7. $\mathrm{temp} = \mathrm{w(e_{ab})}$, $\mathrm{E} = \mathrm{E} - \{\mathrm{e_{ab}}\} \cup \{\mathrm{e_{cb}}\}$, $\mathrm{w(e_{cb})} = \mathrm{w(e_{cb})} + \mathrm{w(e_{ab})}$
8. apply the optimal linear ordering algorithm on the subtree rooted at node c
9. Current_average_access_time= average access time of the output broadcast program from Step 8
10. If (Current_average_access_time < Previous_average_access_time)
11. replace the corresponding broadcast program in *list* by the output broadcast program from Step 8
12. Previous_average_access_time = Current_average_access_time
13. Else
14. $\mathrm{E} = \mathrm{E} \cup \{\mathrm{e_{ab}}\} - \{\mathrm{e_{cb}}\}$, $\mathrm{w(e_{ab})} = \mathrm{temp}$
15. Else if it falls in (Case I.b or Case II) and there exist some nodes between a and b
16. $\mathrm{temp} = \mathrm{w(e_{ab})}$, $\mathrm{w(e_{ab})} = \mathrm{w(e_{cb})} + \mathrm{w(e_{ab})}$
17. apply the optimal linear ordering algorithm on the subtree rooted at node a
18. Current_average_access_time=average access time of the output broadcast program from Step 17
19. If (Current_average_access_time < Previous_average_access_time)
20. replace the corresponding broadcast program in *list* by the output broadcast program from Step 17
21. Previous_average_access_time = Current_average_access_time.
22. Else
23. $\mathrm{w(e_{ab})} = \mathrm{temp}$
24. output *list*

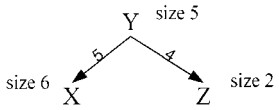

Fig. 13. The construction of $\mathrm{G_f}$ for the access forest in Fig. 12.

4.3 Merging Access Trees

As mentioned in Section 2, the output of the maximum branching algorithm is an access forest. Therefore, in addition to scheduling each access tree, we also need to merge the scheduling results of the access trees. In the access forest, if there is no removed edge between the access trees, we can simply concatenate the scheduling results of the access trees for the broadcast. If there exist some removed edges between the access trees, i.e., the $\mathrm{RE}_{\mathrm{inter}}$ is not empty, the edges in $\mathrm{RE}_{\mathrm{inter}}$ are used to merge the results of access tree scheduling.

In order to schedule an access forest, we consider each access tree as a vertex. An access graph $\mathrm{G_f}$ with each vertex representing an access tree can then be generated by creating edges between the vertices. For an access tree X, define a membership function $In_X(i)$, which returns 1 if a vertex i is in X. Otherwise, it returns 0. For two access trees X and Y, if

$$\sum_{e_{ij} \in RE_{\mathrm{inter}}} In_X(i) \times In_Y(j) \times w(e_{ij})$$

is not zero, an edge $\mathrm{e_{XY}}$ is created in $\mathrm{G_f}$ with

$$w(e_{XY}) = \sum_{e_{ij} \in RE_{\mathrm{inter}}} In_X(i) \times In_Y(j) \times w(e_{ij}).$$

Moreover, the size of the vertex which represents the access tree X $(\mathrm{V_X}, \mathrm{E_X})$ is set to $\sum_{i \in V_x} \|i\|$. Fig. 13 shows the $\mathrm{G_f}$ constructed from the access forest in Fig. 12. After creating $\mathrm{G_f}$, the process shown in Fig. 7 is used to schedule $\mathrm{G_f}$. The process will repeat until no edge can be created for a new $\mathrm{G_f}$. The merging algorithm is presented as follows:

Merging Algorithm

1. Create $\mathrm{G_f}$
2. If there is no edge in $\mathrm{G_f}$
3. Concatenate the scheduling results of the access trees represented by the vertices in $\mathrm{G_f}$
4. Stop
5. Apply the Access Graph RefinementAlgorithm to $\mathrm{G_f}$
6. Run the maximum branching algorithm
7. For each access tree *t* in the access forest
8. Apply the Scheduling Algorithm on t
9. Goto Step 1

According to the Merging algorithm, the broadcast program of the access forest can be determined. Therefore, we only need to concatenate the OLO of each access tree according to the broadcast program determined by the Merging Algorithm to obtain a broadcast program. Fig. 14

YZX: gkhlmjiedfabc

Fig. 14. The merging result and the final broadcast program for the access graph in Fig. 6a.

shows the merging result of the G_f shown in Fig. 13 and the final broadcast program based on the merging result.

5 Performance Evaluation

The performance evaluation consists of two parts, one for the complete approach and the other for the scheduling algorithm only. The simulation is run on a Pentium III 700 processor with 512k cache and 128M memory. In the simulation, we assume that the average access time denotes the number of broadcast buckets needed to be accessed for downloading the set of desired data objects. To evaluate the performance of the complete approach, a set of experiments is performed based on various sets of query patterns. We compare the average access time for the broadcast program generated by the approach with the lower bound on the average access time for the optimal broadcast program of the query patterns. The lower bound on the average access time for the optimal broadcast program of the query patterns is derived in Section 5.1.1.

To evaluate the performance of the proposed scheduling algorithm, a set of experiments is performed by generating different kinds of access graphs. We compare the performance of the proposed algorithm with the one proposed in [6]. In [6], a scheduling algorithm called *PartiallyLinearOrder* is proposed to schedule a weighted acyclic access graph. In the algorithm, the edge with the largest weight is removed from the access graph and the vertices connected by the edge are merged into one vertex, named *multi_vertices*, where the order of the vertices in the multi_vertices is determined by an equation. The process is repeated until all edges are removed and the final multi_vertices is the broadcast program. Notice that our scheduling algorithm can schedule any directed weighted access graph. Moreover, our approach can deal with the variation in data object sizes. In addition to comparing with the result of the PartiallyLinearOrder, the average access time of our approach is also compared with the lower bound on the average access time for the optimal broadcast program of the access graph. The lower bound on the average access time for the optimal broadcast program of the access graph is derived in Section 5.1.2.

5.1 The Derivation of Lower Bounds

5.1.1 The Lower Bound on the Average Access Time for a Set of Query Patterns

The lower bound on the average access time for the optimal broadcast program of the query patterns is derived as follows:

The notations used in the following equations are defined first.

SA_j: The SA set of the jth query pattern.

JA_j: The JA set of the jth query pattern.

PA_j: The PA set of the jth query pattern.

$|SA_j|$: The number of data objects in SA_j.

$d^i_{SA_j-JA_j}$: The number of data buckets needed to reach the data objects in JA_j after all the data objects in SA_j are downloaded when the ith data object in SA_j is the first downloaded data object. The meaning of $d^i_{SA_j-JA_j}$ is shown in Fig. 15.

$d^i_{JA_j-PA_j}$: The number of data buckets needed to reach the data objects in PA_j after all the data objects in SA_j and JA_j are downloaded when the ith data object in SA_j is the first downloaded data object. The meaning of $d^i_{JA_j-PA_j}$ is shown in Fig. 15.

$d^i_{SA_j}$: The number of data buckets needed to download all the data objects in SA_j when the ith data object in SA_j is the first downloaded data object. The meaning of $d^i_{SA_j}$ is shown in Fig. 16.

$d^i_{JA_j}$: The number of data buckets needed to download all the data objects in JA_j when the ith data object in SA_j is the first downloaded data object. The meaning of $d^i_{JA_j}$ is shown in Fig. 16.

$d^i_{PA_j}$: The number of data buckets needed to download all the data objects in PA_j when the ith data object in SA_j is the first downloaded data object. The meaning of $d^i_{PA_j}$ is shown in Fig. 16.

$\|SA_j\|$: The number of data buckets needed to broadcast the data objects in SA_j, i.e., $\sum_{y\in SA_j}\|y\|$.

$\|JA_j\|$: The number of data buckets needed to broadcast the data objects in JA_j, i.e., $\sum_{y\in JA_j}\|y\|$.

$\|PA_j\|$: The number of data buckets needed to broadcast the data objects in PA_j, i.e., $\sum_{y\in PA_j}\|y\|$.

w_j: The access frequency of the jth query pattern.

n: The number of query patterns.

W: The total access frequencies of the query patterns, i.e., $\sum_{j=1}^{n} w_j$.

For a query pattern [SA, JA, PA], data objects in SA should be accessed before data objects in JA and data objects in PA are accessed last. Among the data objects in SA, JA, or PA, there is no access order constraint. There are six steps to access the data objects in a query pattern. First, a client tunes in the broadcast channel and waits to access a data object in SA. Notice that the first data object to be accessed can be any data object in SA. For example, referring to Fig. 15, the first data object to be accessed can be data object a or data object b. Second, all the data objects in SA are downloaded. Third, it waits to access a data object in JA. Fourth, all the data objects in JA are downloaded. Fifth, it waits to access a data object in PA. Finally, all the data objects in PA are downloaded. According to the above steps, the average access time AT_j for accessing all the data objects in query pattern j can be derived:

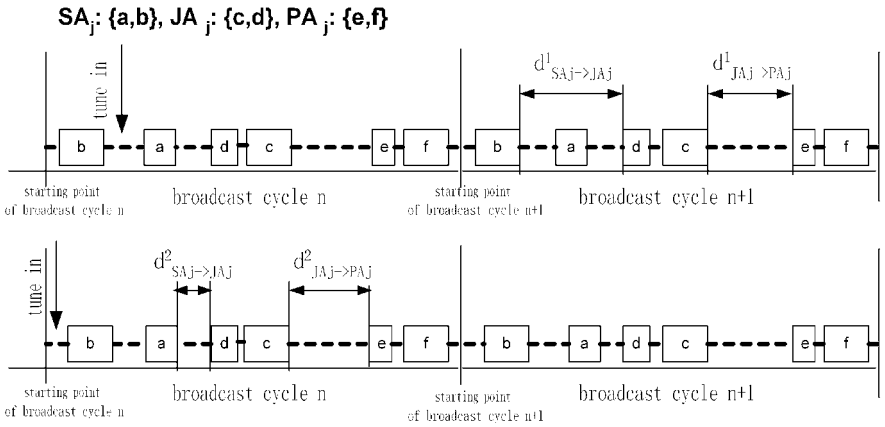

Fig. 15. The definitions of $d^i_{PA_j \to JA_j}$ and $d^i_{JA_j \to PA_j}$.

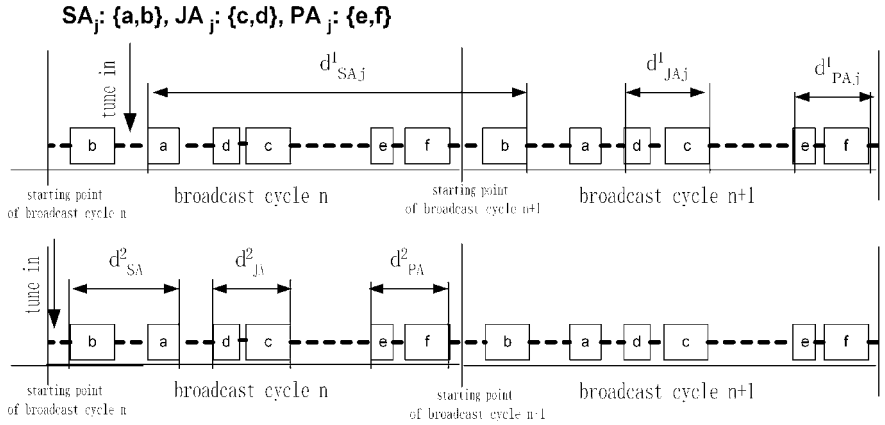

Fig. 16. The definitions of $d^i_{SA_j}$, $d^i_{JA_j}$, and $d^i_{PA_j}$.

$$
\begin{aligned}
&(1/b) \times \Bigg(\sum_{k=0}^{x_1} (k + d^1_{SA_j} + d^1_{SA_j \to JA_j} + d^1_{JA_j} + d^1_{JA_j \to PA_j} + d^1_{PA_j}) \\
&+ \ldots + \sum_{k=0}^{x_{|SA_j|}} (k + d^{|SA_j|}_{SA_j} + d^{|SA_j|}_{SA_j \to JA_j} + d^{|SA_j|}_{JA_j} \\
&\quad + d^{|SA_j|}_{JA_j \to PA_j} + d^{|SA_j|}_{PA_j}) \Bigg) \\
&= (1/b) \times \Bigg(\sum_{i=1}^{|SA_j|} \sum_{k=0}^{x_i} k + \sum_{i=1}^{|SA_j|} (d^i_{SA_j} + d^i_{SA_j \to JA_j} + d^i_{JA_j} \\
&\quad + d^i_{JA_j \to PA_j} + d^i_{PA_j}) \Bigg) \\
&= (1/b) \times \Bigg(\sum_{i=1}^{|SA_j|} (1/2) \times (x_i^2 + x_i) + \sum_{i=1}^{|SA_j|} (d^i_{SA_j} + d^i_{SA_j \to JA_j} \\
&\quad + d^i_{JA_j} + d^i_{JA_j \to PA_j} + d^i_{PA_j}) \Bigg).
\end{aligned}
$$

In the equation, x_i denotes the maximal offset from the tune-in bucket to the first bucket of the first accessed data object in $\mathrm{SA_j}$, where the first accessed data object is the ith data object in $\mathrm{SA_j}$. Therefore, $\sum_{i=1}^{|SA_j|} x_i$ equals $\mathrm{b} - |\mathrm{SA_j}|$.

The average access time for all the query patterns is:

$$(1/W) \times \sum_{j=1}^{n} (w_j \times AT_j).$$

Consider $\mathrm{AT_j}$. $d^i_{RA_j}$ is minimal when all the data objects in SA are allocated adjacently. In this case, $d^i_{JA_j}$ equals $\|\mathrm{SA_j}\|$. Similarly, the minimal values of $d^i_{JA_j}$ and $d^i_{PA_j}$ are $\|\mathrm{JA_j}\|$ and $\|\mathrm{PA_j}\|$, respectively. Moreover, the minimal value of $d^i_{SA_j - JA_j}$ and $d^i_{JA_j - PA_j}$ is zero. Therefore,

$$
\begin{aligned}
AT_j(1/b) \times \Bigg(&\sum_{i=1}^{|SA_j|} (1/2) \times (x_i^2 + x_i) + |SA_j| \\
&\times (\|SA_j\| + \|JA_j\| + \|PA_j\|) \Bigg).
\end{aligned}
$$

Since $\sum_{i=1}^{|SA_j|} (1/2) \times (x_i^2 + x_i)$ equals

$$(1/2) \times \left(\sum_{i=1}^{|SA_j|} x_i^2 + \sum_{i=1}^{|SA_j|} x_i \right)$$

and $\sum_{i=1}^{|SA_j|} x_i$ equals $\mathrm{b} - |\mathrm{SA_j}|$, we need to derive the lower bound of $\sum_{i=1}^{|SA_j|} x_i^2$. We employ the Cauchy-Schwarz inequality [15] for this purpose. The *Cauchy-Schwarz* says that the inner product of two vectors $\vec{a}$ and $\vec{b}$, i.e., $\vec{a} \bullet \vec{b}$, is equal to or smaller than $|\vec{a}| \times |\vec{b}|$. Let $\vec{a} = (x_1, x_2, \ldots, x_{|SA_j|})$ and $\vec{b} = (1, 1, \ldots, 1)$. $\vec{a} \bullet \vec{b}$ is equal to $\sum_{i=1}^{|SA_j|} x_i$ and $|\vec{a}| \times |\vec{b}|$ is equal to

$$\sqrt{\sum_{i=1}^{|SA_j|} x_i^2} \times \sqrt{|SA_j|}.$$

We have

TABLE 1
Parameter Setting for Query Pattern Generation

Parameters	Default value	Ranges
Number of data objects	400	100 – 700
Data object size	Zipf(20, 80, 0.4)	Zipf(20, 80, θ), θ = 0, 0.2, 0.4, 0.6, 0.8, 1
Access frequency of a query pattern	Zif(10,100,0.4)	Zipf(10,100,θ), θ = 0, 0.2, 0.4, 0.6, 0.8, 1
Num ber of query patterns	5000	100500, 1000, 3000, 5000, 7000, 9000, 11000
Number of data objects in a query pattern	Zipf(3,12,0.4)	Zipf(3, 12, θ), θ = 0, 0.2, 0.4, 0.6, 0.8, 1

$$\sum_{i=1}^{|SA_j|} x_i \le \sqrt{\sum_{i=1}^{|SA_j|} x_i^2} \times \sqrt{|SA_j|}$$
$$\Rightarrow \left(\sum_{i=1}^{|SA_j|} x_i\right)^2 \le |SA_j| \times \sum_{i=1}^{|SA_j|} x_i^2$$
$$\Rightarrow (b - |SAj|)^2/|SAj| \le \sum_{i=1}^{|SA_j|} x_i^2.$$

Therefore,

$$AT_j(1/b) \times ((1/2) \times ((b - |SA_j|)^2/|SA_j| + b - |SA_j|) + |SA_j| \times (||SA_j|| + ||JA_j|| + ||PA_j||)),$$

i.e.,

$$AT_j(1/2) \times (b/|SA_j| - 1) + (|SA_j|/b) \times (||SA_j|| + ||JA_j|| + ||PA_j||).$$

The average access time for all the query patterns is

$$(1/W) \times \sum_{j=1}^{n} (w_j \times AT_j),$$

which is equal to or larger than

$$(1/W) \times \sum_{j=1}^{n} w_j \times ((1/2) \times (b/|SA_j| - 1) + (|SA_j|/b) \times (||SA_j|| + ||JA_j|| + ||PA_j||)).$$

The lower bound on the average access time for all the query patterns is gotten.

5.1.2 The Lower Bound on the Average Access Time for an Access Graph

The lower bound on the average access time for an access graph is derived as follows:

Referring to Section 2, the average access time is

$$(1/b) \times \sum_{k=0}^{b-1} \sum_{e_{ij} \in E} \left((w(e_{ij}) / \sum_{e_{ij} \in E} w(e_{ij})) \times (k + r_{i-j}) \right).$$

$r_{i \to j}$ is minimal when data object i is allocated right before data object j. In this case, $r_{i \to j}$ is equal to |i| + |j|. Therefore, the lower bound of the average access time is

$$(1/b) \times \sum_{k=0}^{b-1} \sum_{e_{ij} \in E} \left((w(e_{ij}) / \sum_{e_{ij} \in E} w(e_{ij})) \times (k + \|i\| + \|j\|) \right)$$
$$= ((b-1)/2) + \left(1/\sum_{e_{ij} \in E} w(e_{ij}) \times \sum_{e_{ij} \in E} (w(e_{ij}) * (\|i\| + \|j\|) \right).$$

5.2 Experiment Setup

The following parameters (some of them are adopted from [6]) are used to generate a set of query patterns and a set of access graphs.

PARAMETERS:

- *Number of data objects*: The number of data objects being broadcast, which is also the number of vertices in the access graph.
- *Data object size*: The size of each data object being broadcast.
- *Out-degree*: The out-degree for each vertex in the access graph.
- *Edge weight*: The weight associated with each edge in the access graph.

The parameter settings for evaluating the complete approach and the scheduling algorithm only are listed in Table 1 and Table 2, respectively. The notation $\mathrm{Zipf(a, b, }\theta\mathrm{)}$ denotes a range of numbers from a to b in the Zipf's distribution [10] with factor θ. Notice that the values generated by Zipf(a, b, 0) are uniformly distributed in [a, b]. Moreover, as θ increases, the probability of generating a large value increases. The value generated by Zipf(a, b, 1) is b. We assume the out-degree of each vertex is among the values of {0, 1, 2, 3}. Also, we use the ratio of the out-degrees to vary the connectivity of the access graph. For example, when the ratio is 6:1:1:1, the probability of the vertex with zero out-degree is six times of that of the vertex with other out-degrees.

5.3 Experimental Results

5.3.1 Evaluating the Performance of the Complete Approach

Fig. 17 shows the effect of the number of data objects. The average access time increases as the number of data objects increases. The smaller the number of data objects is, the better the performance of our approach is. For example, when the number of data objects is 100 and 700, the ratio of the average access time of our approach to the lower bound

TABLE 2
Parameter Settings for Access Graph Generation

Parameters	Default value	Ranges
Number of data objects	400	100 – 700
Data object size	Zipf(20, 80, 1)	Zipf(20, 80,θ), θ = 0, 0.2, 0.4, 0.6, 0.8, 1
Ratio of out-degrees in[0,1,2,3]	1:1:6:1	1:1:1:1, 6:1:1:1, 1:6:1:1, 1:1:6:1, 1:1:1:6
Edge weight	Zipf(10, 100, 0.4)	Zipf(10, 100,θ), θ = 0, 0.2, 0.4, 0.6, 0.8, 1

on the average access time of the optimal broadcast program is 2.13 and 2.30, respectively. In our approach, the access graph is first constructed to represent the query patterns, then the scheduling algorithm is applied on the access graph to get the broadcast program. As the number of data objects increases, the performance of the scheduling algorithm degrades (the performance of the scheduling algorithm will be separately discussed in Section 5.3.2). Moreover, as the number of data objects increases, the number of the data objects in the query patterns whose access order should be determined by MIW also increases. However, the ratio of the average access time of our approach to the lower bound on the average access time of the optimal broadcast program remains about the same. In the simulation, the ratios are 2.13, 2.21, 2.26, 2.28, 2.29, 2.29, and 2.30 when the number of data objects are 100, 200, 300, 400, 500, 600, and 700, respectively.

The effect of the data object sizes is shown in Fig. 18. When θ equals zero, the data object sizes are uniformly distributed between [20, 80]. Moreover, as the value of θ increases, the probability of generating larger data objects increases. Therefore, the larger the value of θ is, the more the number of data buckets in a broadcast cycle is needed, i.e., which lengthens the average access time. The ratio of the average access time of our approach to the lower bound on the average access time of the optimal broadcast program is invariant to the data object size distribution.

Fig. 19 shows the effect of the access frequency of query patterns. As shown in the figure, the ratio of the average access time of our approach to the lower bound on the average access time of the optimal broadcast program is invariant to the access frequency distribution. The effect of the number of query patterns is shown in Fig. 20. As shown in the result, when the number of query patterns is small, say 100, the ratio of the average access time of our approach to the lower bound on the average access time of the optimal broadcast program is 2.54, which is not as good as the performance shown in Fig. 18. The reason is that, when the number of query patterns is small, the number of data objects in the query patterns whose access order should be determined by MIW increases.

Fig. 21 shows the effect of the number of data objects in a query pattern. As shown in the result, in our approach, the average access time increases as the number of data objects in a query pattern increases. However, the lower bound on the average access time of the optimal broadcast program decreases as the number of data objects in a query pattern increases. The reason is that, as the number of data objects in a query pattern increases, the number of data objects in SA ($|\mathrm{SA}|$) increase. Therefore, $(\mathrm{b} - |\mathrm{SA_j}|)^2/|\mathrm{SA_j}|$ decreases. Referring to Section 5.1.2, the minimal value of $\sum_{i=1}^{|SA_j|} x_i^2$ occurs when

$$\sum_{i=1}^{|SA_j|} x_i = \sqrt{\sum_{i=1}^{|SA_j|} x_i^2} \times \sqrt{|SA_j|}.$$

According to the Cauchy-Schwarz inequality, it means that the direction of $(x_1, x_2, \ldots, x_{n_j})$ and $(1, 1, \ldots 1)$ are the same. That is, $x_1 = x_2 = \ldots = x_{n_j}$, which means that the data objects in SA are uniformly allocated in the broadcast program. However, when considering $d^i_{SA_j}$, the minimal value occurs when the data objects in SA are allocated adjacently, i.e., $\|\mathrm{SA_j}\|$. These two conditions cannot be satisfied at the same time. Moreover, as the number of data objects in a query pattern increases, the time needed to access all the data objects in a query pattern increases. Therefore, in our approach, the average access time

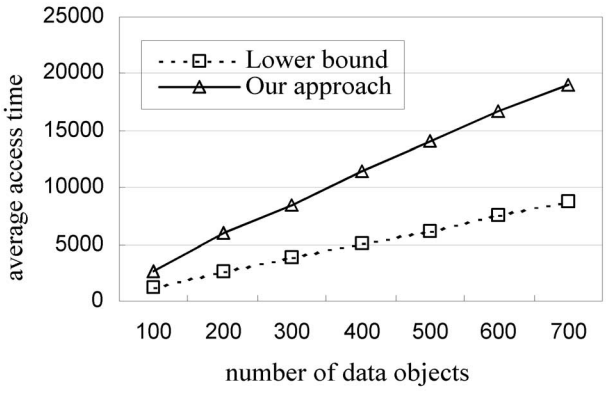

Fig. 17. Effect of the number of data objects.

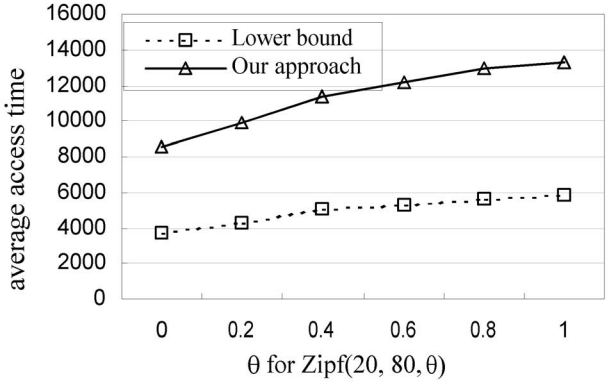

Fig. 18. Effect of data object size.

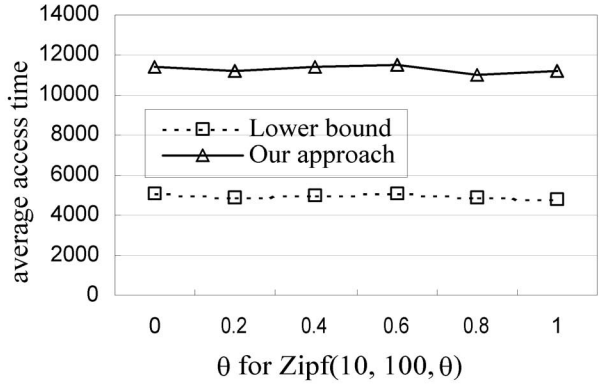

Fig. 19. Effect of query pattern frequency.

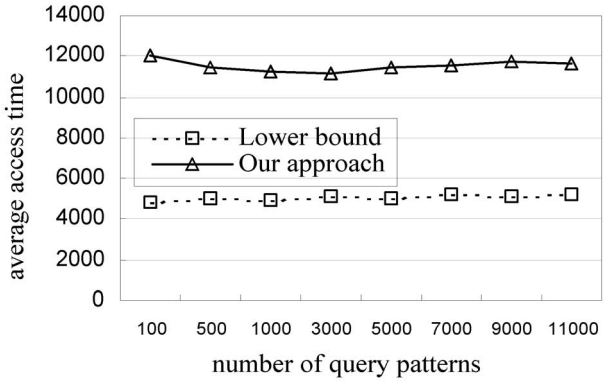

Fig. 20. Effect of the number of query patterns.

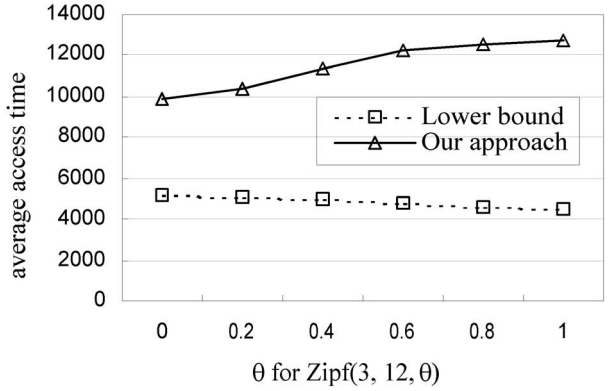

Fig. 21. Effect of the number of data objects in a query pattern.

increases as the number of data objects in a query pattern increases.

5.3.2 Evaluating the Performance of the Scheduling Algorithm

Fig. 22 shows the effect of the number of data objects. The average access time increases as the number of data objects increases. Moreover, as the number of data objects increases, our approach outperforms PartiallyLinearOrder. The reason is that, in our approach, the optimal linear ordering algorithm is first used to determine the major order of the data objects. After determining the major order of the data objects, the information kept in $\mathrm{RE}_{\mathrm{intra}}$ is used to adjust the order to get a better average access time. Therefore, our approach has a global view of the

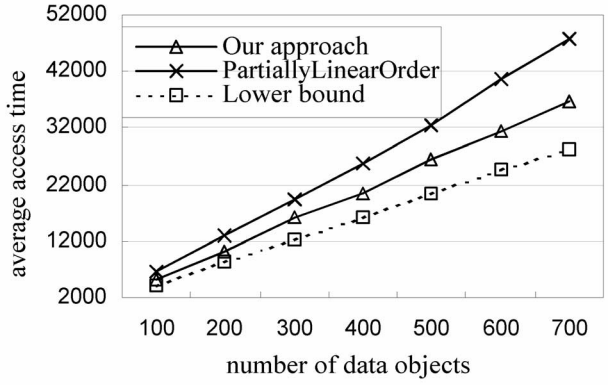

Fig. 22. Effect of the number of data objects.

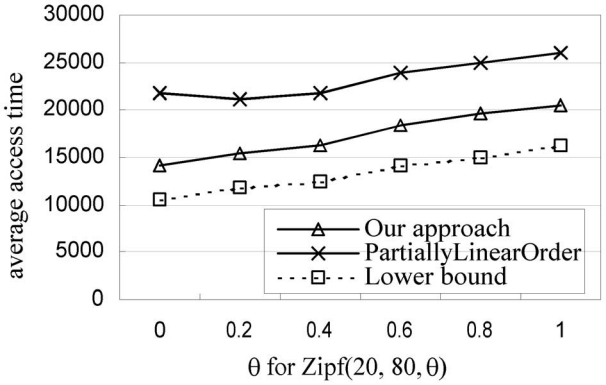

Fig. 23. Effect of data object size.

relationship among data objects. Moreover, when the number of data objects is 700, the ratio of the average access time of our approach to the lower bound on the average access time of the optimal broadcast program is $36,798/28,341 \cong 1.3$, which is a good approximation for solving the scheduling problem.

The effect of the data object sizes is shown in Fig. 23. Our approach outperforms PartiallyLinearOrder, especially when the value of θ is small. The reason is that PartiallyLinearOrder does not take the size of data objects into account. When θ equals zero, the data object sizes are uniformly distributed between [20, 80]. Moreover, as the value of θ increases, the probability of generating larger data objects increases. Therefore, the smaller the value of θ is, the more random the data size distribution is. Partially LinearOrder is not suitable to deal with the variation in data object sizes.

Fig. 24 shows the effect of the ratio of out-degrees. The average access time increases as the number of out-degrees increases. Moreover, as the number of out-degrees increases, our approach outperforms PartiallyLinearOrder. The reason is that, as the number of out-degrees increases, the complexity of the access graph increases. To schedule a complex access graph, an algorithm with a global view will perform better. As mentioned in the previous discussion, our approach has a global view of the relationship among data objects. Therefore, our approach outperforms Partially LinearOrder. Moreover, when the ratio of out-degrees is 6:1:1:1 or 1:6:1:1, the value of the average access time of our approach over the lower bound on the average access time

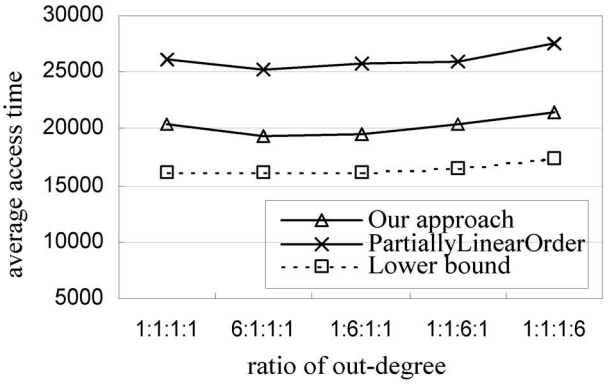

Fig. 24. Effect of the ratio of out-degrees.

of optimal broadcast program is 1.2, which approximates the optimal broadcast program very well. The reason is that, when the access graph is simpler, the number of edges removed by applying the maximum branching algorithm is reduced. The effect of the edge weight is shown in Fig. 25. As shown in the result, the average access time is invariant to the distribution of the edge weights. Moreover, our approach outperforms PartiallyLinearOrder.

6 Conclusion

The data allocation problem on the disk storage has been widely studied in the past. As the application of data broadcast in the mobile environment becomes popular, the issue of data allocation on the broadcast channel for reducing the access latency receives much attention. In this paper, the database broadcast issues are discussed and the idea of the access graph is introduced to represent the data objects with a certain relationship. Moreover, heuristics are proposed to determine the broadcast order for data objects whose relationship is represented by an access graph. This problem can be proven to be NP-complete. We propose a heuristic to solve the problem based on the techniques of solving two well-known problems, the maximum branching problem and optimal linear ordering problem. We transform the access graph to a set of access trees, each of which can be arranged into an optimal broadcast order. Then, we merge these broadcast orders to form the final result. We take the effect of each removed edge from the access graph into consideration, which makes our approach more effective. Our proposed algorithm can deal with any access graph with different sizes of data objects. Experiments show that our approach has good performance. In the future, we will consider the data allocation problem on multiple broadcast channels and the issue of using data replication to increase the availability of popular data objects.

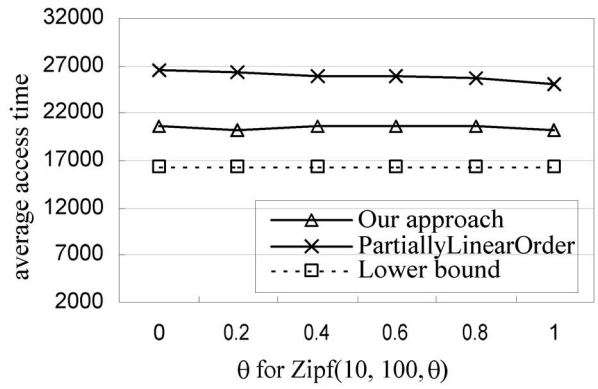

Fig. 25. Effect of edge weight.

Acknowledgments

The authors would like to thank the anonymous referees for their helpful suggestions. This research was partially supported by the National Science Council of the Republic of China under Contract No. NSC 91-2213-E-259-002.

References

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

[2] D. Aksoy and M.J. Franklin, "Scheduling for Large-Scale On-Demand Data Broadcasting," *Proc. IEEE INFOCOM Conf.*, pp. 651-659, 1998.

[3] D. Adolphson and T.C. Hu, "Optimal Linear Ordering," *SIAM J. Applied Math.*, vol. 25, pp. 403-423, 1973.

[4] A. Bar-Noy, J. Naor, and B. Schieber, "Pushing Dependent Data in Clients-Providers-Servers Systems," *Proc. MOBICOM Conf.*, pp. 222-230, 2000.

[5] A. Bar-Noy and Y. Shilo, "Optimal Broadcasting of Two Files over an Asymmetric Channel," *Proc. IEEE INFOCOM Conf.*, pp. 267-274, 1999.

[6] 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, pp. 69-94, July 1999.

[7] Y.D. Chung and M.-H. Kim, "QEM: A Scheduling Method for Wireless Broadcast Data," *Proc. Sixth Int'l Conf. Database Systems for Advanced Applications (DASFAA)*, pp. 135-142, Apr. 1999.

[8] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman Publishing, 1976.

[9] V. Gondhalekar, R. Jain, and J. Werth, "Scheduling on Airdisks: Efficient Access to Personalized Information Services via Periodic Wireless Data Broadcast," *Proc. Int'l Conf. Comm. (ICC)*, pp. 1276-1280, 1997.

[10] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P.J. Weinberger, "Quickly Generating Billion-Record Synthetic Databases," *Proc. ACM SIGMOD Conf.*, pp. 243-252, May 1994.

[11] A.R. Hurson, Y.C. Chehadeh, and J. Hannan, "Object Organization on Parallel Broadcast Channels in a Global Information Sharing Environment," *Proc. IEEE Int'l Conf. Performance, Computing, and Comm.*, pp. 347-353, 2000.

[12] G. Herman, G. Gopal, K.C. Lee, and A. Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems," *ACM SIGMOD Record*, pp. 97-103, 1987.

[13] C.-H. Hsu, G. Lee, and A.L.P. Chen, "A Near Optimal Algorithm for Generating Broadcast Programs on Multiple Channels," *Proc. ACM 10th Int'l Conf. Information and Knowledge Management*, pp. 303-309, 2001.

[14] C.-H. Hsu, G. Lee, and A.L.P. Chen, "Index and Data Allocation on Multiple Broadcast Channels Considering Data Access Frequencies," *Proc. Int'l Conf. Mobile Data Management*, pp. 87-93, 2002.

[15] K. Hoffman and R. Kunze, *LINEAR ALGEBRA*. Prentice Hall, 1971.

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

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

[18] S.C. Lo and A.L.P. Chen, "Optimal Index and Data Allocation in Multiple Broadcast Channels," *Proc. 16th IEEE Int'l Conf. Data Eng.*, pp. 293-302, Feb. 2000.

[19] S.C. Lo and A.L.P. Chen, "An Adaptive Access Method for Broadcast Data under an Error-Prone Mobile Environment," *IEEE Trans. Knowledge and Data Eng.*, vol. 12, no. 4, pp. 609-620, July/Aug. 2000.
[20] G. Lee, S.-C. Lo, and A.L.P. Chen, "Data Allocation on Wireless Broadcast Channels for Efficient Query Processing," technical report, http://mckm.csie.ndhu.edu.tw/lee_research.htm, 2002.
[21] G. Lee and S.-C. Lo, "Broadcast Data Allocation for Efficient Access of Multiple Data Items in Mobile Environments," *ACM Mobile Networks and Applications (MONET)*, to appear.
[22] G. Lee, M.-S. Yeh, S.-C. Lo, and A.L.P. Chen, "A Strategy for Efficient Access of Multiple Data Items in Mobile Environments," *Proc. Int'l Conf. Mobile Data Management*, pp. 71-78, 2002.
[23] 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.
[24] A. Si and H.V. Leong, "Query Optimization for Broadcast Database," *Data and Knowledge Eng.*, vol. 29, no. 3, pp. 351-380, Mar. 1999.
[25] K. Thulasiraman and M.N.S. Swamy, *Graphs: Theory and Algorithms*. Wiley-Interscience, 1992.
[26] N. Vaidya and S. Hameed, "Scheduling Data Broadcast in Asymmetric Communication Environments," *ACM/Baltzer Wireless Networks*, vol. 5, no. 3, pp. 171-182, 1999.
[27] K.H. Yeung and T.S. Yum, "Selective Broadcast Data Distribution Systems," *IEEE Trans. Computers*, vol. 46, no. 1, pp. 100-104, Jan. 1997.

Guanling Lee received the BS, MS, and PhD degrees, all in computer science, from National Tsing Hua University, Taiwan, Republic of China, in 1995, 1997, and 2001, respectively. She joined National Dong Hua University, Taiwan, as an assistant professor in the Department of Computer Science and Information Engineering in August 2001. Her research interests include location management in mobile environments, data scheduling on wireless channels, and data mining.

Shou-Chih Lo received the BS degree in computer science from National Chiao Tung University, Taiwan, in 1993, and the PhD degree in computer science from National Tsing Hua University, Taiwan, in 2000. He is now with the Computer & Communication Research Center at National Tsing Hwa University, Taiwan, as a postdoctoral fellow. His current research interests are in the area of mobile and wireless Internet, with emphasis on mobility management, interworking operation, and MAC protocols with QoS guarantee. He also works on problems related to index and data allocation on broadcast channels. Dr. Lo received the Best Thesis Award from the Chinese Institute of Information & Computer Machinery in 2000.

Arbee L.P. Chen received the BS degree in computer science from National Chiao-Tung University, Taiwan, Republic of China, in 1977, and the PhD degree in computer engineering from the University of Southern California in 1984. He joined National Tsing Hua University (NTHU), Taiwan, as a National Science Council (NSC) sponsored visiting specialist in August 1990 and became a professor in the Department of Computer Science in 1991. In August 2001, he took a leave from NTHU and assumed the position of the chairman of the Department of Computer Science and Information at National Dong Hwa University, Hualien, Taiwan. He was a member of the technical staff at Bell Communications Research, New Jersey, from 1987 to 1990, an adjunct associate professor in the Department of Electrical Engineering and Computer Science, Polytechnic University, New York, and a research scientist at Unisys, California, from 1985 to 1986. His current research interests include multimedia databases, data mining, and mobile computing. Dr. Chen has organized the 1995 IEEE Data Engineering Conference and 1999 International Conference on Database Systems for Advanced Applications (DASFAA) in Taiwan. He is an editor of several international journals, including *World Wide Web: Internet and Web Information Systems* (Kluwer Academic Publishers). He has been a recipient of the National Science Council Distinguished Research Award since 1996. He is a senior member of the IEEE.