An adaptable vertical partitioning method in distributed systems

Jin Hyun Son a,*, Myoung Ho Kim b

a Department of Computer Science and Engineering, Hanyang University, 1271, Sa I-dong, Ansan-si, Gyeonggi-do 426-791, South Korea
b Division of Computer Science, Korea Advanced Institute of Science and Technology, 373-1, Kusong-dong, Yusong-gu, Taejon 305-701, South Korea

Received 18 February 2002; received in revised form 12 June 2002; accepted 14 April 2003
Available online 30 December 2003

Abstract

Vertical partitioning is a process of generating the fragments, each of which is composed of attributes with high affinity. The concept of vertical partitioning has been applied to many research areas, especially databases and distributed systems, in order to improve the performance of query execution and system throughput. However, most previous approaches have focused their attention on generating an optimal partitioning without regard to the number of fragments finally generated, which is called best-fit vertical partitioning in this paper. On the other hand, there are some cases that a certain number of fragments are required to be generated by vertical partitioning, called n-way vertical partitioning in this paper. The n-way vertical partitioning problem has not fully investigated.

In this paper, we propose an adaptable vertical partitioning method that can support both best-fit and n-way vertical partitioning. In addition, we present several experimental results to clarify the validness of the proposed algorithm.

© 2003 Published by Elsevier Inc.

1. Introduction

The concept of data distribution in distributed systems has been applied to achieve the high performance distributed processing, especially data-centric information processing. Data distribution can generally be accomplished by two consecutive steps: data partitioning and data allocation. First, data partitioning, sometimes called data clustering, divides a database into several fragments. And then, data allocation to follow after data partitioning involves finding the optimal distribution of the fragments into sites while satisfying response time, storage, or processing constraints.

Data partitioning technologies can be applied to the areas where the match between data and queries (or transactions) affects performance. These areas include partitioning of individual files in centralized environments, data distribution in distributed databases, dividing data among different levels of memory hierarchies, and so on (Navathe and Ra, 1989). Data partitioning can be categorized into vertical and horizontal partitioning according to the target objects (Ozsu and Valduriez, 1999). Vertical partitioning decomposes data attributes into groups that are composed of some attributes with high affinity, while horizontal partitioning divides data tuples into groups (Ozsu and Valduriez, 1999; Navathe et al., 1984). In addition, hybrid partitioning mixes the two partitioning methods, i.e., vertical partitioning followed by horizontal partitioning or vice versa (Ozsu and Valduriez, 1999). These partitioning schemes are primarily based on data access (i.e., query) patterns. In this paper, we focus on the vertical partitioning problem in distributed systems.

Vertical partitioning in distributed systems can improve system throughput as well as the performance of query execution for queries running on parts of data attributes. The high performance of query execution can usually be achieved by two ways. First, the frequency of queries to access different data fragments should be decreased. Second, the amount of unneeded information brought into memory from secondary storage should be reduced (Cornell and Yu, 1990; Chu and Leong, 1993;
Navathe et al., 1984; Navathe and Ra, 1989). In a distributed system environment, the former is more important because it could reduce the inter-site data transmission, and to some extent it can also achieve the effect of the latter. In addition, with appropriate allocation of vertically partitioned fragments, overall system throughput can be enhanced by means of parallel query processing to the fragments located in different sites (Ozsu and Valduriez, 1999; Balasundaram et al., 1990; Sacca and Wiederhold, 1985; Bellatreche and Simonet, 1996; Hwang and Yang, 1998).

1.1. Motivation

Two different vertical partitioning can be performed into two directions according to the restriction on the number of fragments finally generated. One is to generate an overall optimal partitioning that minimizes the processing cost of queries without restriction on the number of fragments generated. We call this kind of vertical partitioning best-fit vertical partitioning in this paper. The other is to generate the specific number of fragments required by the user, which is generally called n-way vertical partitioning. While there have been much work on best-fit vertical partitioning (Balasundaram et al., 1990; Sacca and Wiederhold, 1985; Cornell and Yu, 1990; Chu and Leong, 1993; Bellatreche and Simonet, 1996; Hwang and Yang, 1998; Navathe et al., 1984; Navathe and Ra, 1989), the n-way vertical partitioning problem has not been sufficiently investigated. Let us consider that the network is composed of k sites and we want to fully and evenly utilize their system resources. To meet this requirement, we need to generate at least k number of fragments to evenly distribute the fragments to k sites, which can be supported by n-way vertical partitioning. The application of n-way vertical partitioning can be extended to the areas of locating computer components, especially computer software processes in distributed systems.

Our motivation for the research in this paper is based on the following three reasons. First, most vertical partitioning approaches proposed so far support only best-fit vertical partitioning. A vertical partitioning method may be more effective and useful if it can support n-way vertical partitioning as well as best-fit vertical partitioning. Second, previous methods such as Navathe et al. (1984) provides n-way vertical partitioning only in certain cases. The binary vertical partitioning (BVP) proposed in Navathe et al. (1984) mentioned the repetitive use of BVP to support n-way vertical partitioning. The BVP algorithm uses an objective function as the partitioning measure, where BVP cannot progress anymore when the value of the objective function is not positive. This means that in many cases the repetitive use of BVP cannot support n-way partitioning. In addition, BVP does not give any guidelines in selecting the fragment that will be further partitioned in each repetitive process. For example, when we want to obtain three data fragments from one data schema, at least two repetitive binary partitioning must be performed as in Fig. 1. Because there are two fragments F1 and F2 as the results of the first binary partitioning and each fragment can be further partitioned independently, we should select one of the two fragments for another binary partition to obtain three fragments. BVP does not give any selection guidelines for this case. Because of that, BVP may generate several alternative partitioning results. That is, BVP cannot always give a deterministic solution for the n-way vertical partitioning problem. In summary, the repetitive use of BVP cannot be a complete solution for n-way vertical partitioning. The last reason for our motivation is that the previous methods do not use the same algorithm to support both best-fit partitioning and n-way partitioning. In other words, some additional operations or algorithms for n-way partitioning are needed. It may, however, be more effective if we can support the two kinds of vertical partitioning in the same algorithm.

In this paper, we propose an efficient and adaptable vertical partitioning method called AVP standing for the adaptable vertical partitioning that can support not only best-fit vertical partitioning but also n-way vertical partitioning in a single algorithm.

1.2. Related work

There are many previous work on vertical partitioning. Because vertical partitioning basically stores together attributes that are frequently accessed together by queries, Hoffer and Severance (1975) measured the affinity between pairs of attributes using the bond energy algorithm (BEA) in McCormick et al. (1972). After that, this affinity measure has been widely used in many related studies.

Hammer and Njamir (1979) proposed two heuristics, the pairwise grouping heuristics and the attribute regrouping heuristics, that can automatically select near-optimal attribute partitions from the attributes of a file, based on the usage pattern of the file and the characteristics of the data in the file. These two heuristics are alternatively applied until no further improvement for
attribute partitioning, beginning with single-attribute partitions and merging them (i.e., a bottom-up approach). In contrast to Hammer and Njamir (1979), Navathe et al. (1984) proposed a cluster algorithm and a binary partitioning algorithm using the bond energy algorithm (BEA), which is a top–down approach. That is, vertical partitioning in Navathe et al. (1984) begins with a group composed of all attributes and splits them. The binary partitioning technique is required to be repeated until no more benefits can be obtained, coming with complex computation of an empirical objective function. And, clustering is also repeated at each iteration after clustering two new affinity matrices corresponding to the newly generated fragments. Navathe et al. (1984) also discussed the n-way vertical partitioning problem, but the proposal could provide exactly n fragments only in certain cases as mentioned in Section 1.1. Navathe and Ra (1989) proposed a new vertical partitioning algorithm which has less computational complexity than Navathe et al. (1984) and generates all meaningful fragments simultaneously. But, Navathe and Ra (1989) does not support n-way partitioning.

Database partitioning and allocation algorithms in a cluster of processors were developed in Sacca and Wiederhold (1985) under the concept that the partitioning and allocation of the database over the processor sites of the network can be a critical aspect of the database design effort. The overall algorithm is composed of a greedy heuristic and a first-fit algorithm. In Hufnagel and Browne (1989), it was observed that the major obstacle to widespread use of object-oriented systems would be that their execution may be intrinsically inefficient due to excessive overhead. To reduce this overhead, Hufnagel and Browne (1989) proposed a vertically partitioned structure for design and implementation of object-oriented systems. On the other hand, Cornell and Yu (1990) and Chu and Leong (1993) considered that the response time of a transaction or query in a relational database system is strongly affected by the number of disk accesses. They insisted that appropriate data partitioning through query analyses would be useful to minimize the number of disk accesses. In line with this, Cornell and Yu (1990) proposed an approach based on an integer programming technique which consists of a query analysis step and a partitioning step. And, Chu and Leong (1993) proposed transaction-based binary partitioning algorithms with an objective function of minimizing the total number of disk accesses, insisting that transactions have more semantic meanings than attributes but all previous research used an attribute as the basic manipulation unit. Even though the results of Cornell and Yu (1990) and Chu and Leong (1993) may be effective in a relational database system, their algorithms are not appropriate to apply to distributed system environments which are common nowadays.

For a long time, the problem of data fragmentation has been recognized for its impact upon the performance of the distributed relational database systems as a whole. But, from gaining popularity of object-oriented databases Bellatreche and Simonet (1996) proposed class fragmentation and allocation schemes in order to minimize data transfer in distributed object database systems with complex attributes and methods. Besides database systems, Hwang and Yang (1998) addressed the necessity of component and data distribution in designing a distributed workflow management system (WFMS).

The remainder of the paper is organized as follows. In Section 2, we mention the basic information and the cost model that are necessary to develop a vertical partitioning method in distributed systems. In Section 3, we propose our vertical partitioning method called AVP. Section 4 gives several experiments to show the efficiency of the proposed method. Finally, we summarize the paper with its contribution in Section 5.

2. Preliminary

2.1. Information requirements

Vertical partitioning is performed for a data schema using the information of data query (or transaction) patterns. A data schema is composed of a set of data attributes which are the target objects of vertical partitioning. If there is no ambiguity, data attributes will be called simply attributes in the following. Vertical partitioning is a process of generating a set of fragments, each of which is composed of attributes that are frequently accessed together by queries. The followings are the information required for developing a new vertical partitioning method in this paper.

- \( R(\alpha_1, \alpha_2, \ldots, \alpha_n) \): a data schema which is composed of \( n \) attributes;
- \( Q = \{ q_1, q_2, \ldots, q_t \} \): a set of data queries that will run on data schema \( R \);
- \( F = \{ f_1, f_2, \ldots, f_t \} \): the frequency \( f_i \) of each query \( q_i \).

In general, there are two kinds of fragmentation in vertical partitioning: non-overlapping and overlapping. The non-overlapping fragmentation generates mutually exclusive fragments except primary key attributes, while the fragments generated by the overlapping fragmentation may commonly share some data attributes. The non-overlapping fragmentation cannot only cover most target applications but easily be extended to support the overlapping fragmentation (Ozsu and Valduriez, 1999). Therefore, we in this paper consider a vertical partitioning method to generate non-overlapping fragments.
2.2. Cost model

As we mentioned in Section 1, the objective of vertical partitioning in distributed systems is to improve both the performance of query execution and system throughput for data queries (i.e., transactions) running on parts of the data schema. First, the high performance of query execution is closely related to minimizing the access cost of data fragments. Because the frequency of accessing different fragments in a query may be a major factor to affect the query execution cost, it is very important to minimize this kind of the frequency for the high performance of query execution. On the other hand, the enhancement of overall system throughput for data queries can be achieved by maximizing the degree of parallel execution. Because data queries may generally run on parts of data attributes, we can improve the degree of parallel execution if we can reduce the frequency of interfered accesses between data queries. Therefore, we can define the cost model that reflects both objectives of vertical partitioning mentioned above.

Objective function: Min\[c \cdot \text{avg}(DF) + \text{avg}(IA)]

where:

- $DF$: the total frequency of accessing different fragments
- $IA$: the total frequency of interfered accesses between data queries
- $DF_{avg}$: the average $DF$ per data query
- $IA_{avg}$: the average $IA$ per data fragment
- $c$: the proportional constant between $DF$ and $IA$

Because the value of $DF$ is related to the performance of data query execution, most previous vertical partitioning methods use it or its variants (such as the total number of disk accesses) as their cost models. On the other hand, the value of $IA$ is connected with system throughput for data queries. One of the purposes of distributed systems is to enable applications and service processes to proceed concurrently without competing for the same resources and to exploit the available computational resources (processors, memory and network capacities) (Coulouris et al., 2001). Hence, $IA$ in the cost model can be said to reflect this characteristic of distributed systems. The proportional constant $c$ between $DF$ and $IA$ may be determined according to the target applications and system environment. Because $DF$ is generally considered as the dominant cost factor in the previous approaches, we should consider the cost model with the value of $c$ greater than 1 when our proposed method is compared with them.

3. Adaptable vertical partitioning

Our adaptable vertical partitioning (AVP) method proposed in this paper supports $n$-way vertical partitioning as well as best-fit vertical partitioning, based on the cost model defined in Section 2.2. AVP is composed of two consecutive phases: constructing a partition tree and then selecting a set of fragments as the output of vertical partitioning. Each phase will be discussed in detail.

3.1. Partition tree

The way to generate data fragments from a data schema can be classified into top–down and bottom–up approaches. In a top–down approach, vertical partitioning begins with a group composed of all data attributes belonging to a data schema and then we proceed down by splitting the group (Navathe et al., 1984). On the other hand, a bottom–up approach begins with groups that are single-attribute fragments and then we proceed up by grouping them (Hammer and Njamir, 1979). Our AVP method proposed in this paper is based on a bottom–up approach. We first begin with single-attribute fragments. And then, we form a new fragment by selecting and merging two fragments of them. This process will be repeated until a fragment composed of all data attributes is made. This kind of a bottom–up approach generates a binary tree as in Fig. 2. Especially, we call it a partition tree, simple PT. As you expect, a PT is constructed from the leaf nodes to the root node. Each node in a PT corresponds to a data fragment, especially, leaf nodes correspond to single-attribute fragments and a root node to a data fragment composed of all data attributes. Let the level of the leaf node in a PT be Step 0 as in Fig. 2. We construct nodes within Step 1 of Fig. 2 by selecting and merging proper two leaf nodes. Other leaf nodes except the merged two leaf nodes are also kept in Step 1 without changes. Hence, when there are $n$ leaf nodes in Step 0, Step 1 has $n – 1$ nodes. This process is repeated until the root node of a PT is made as in Fig. 2. As a result, we can construct a PT with height $n$ from $n$ data attributes. On the other hand, the table in Fig. 2 describes the access pattern and frequency of each data query. That is, query $q_1$ accesses attribute $a_2$, $a_3$, and $a_4$ with frequency 15.

In the process of constructing a PT, we need to merge two nodes (i.e., fragments) for each step. Here, a scheme that can generate a new node by selecting and merging two nodes is required. When two nodes (fragments) are merged into a node (fragment), we can notice the effects of two cost factors mentioned in the cost model of the paper. In a merged fragment compared to previous two fragments, the total frequency for data queries to access different fragments (i.e., $DF$) will be decreased while the frequency of newly interfered accesses between data queries (i.e., $IA$) will be created. Consider that two fragments $A$ and $B$ are merged into a fragment $C$, and some data queries access both fragment $A$ and $B$. After merging the fragments, these data queries have only to
access the merged fragment $C$, resulting in decreasing $\#DF$. Let the data queries to access only fragment $A$ be $q_i^A$ and the data queries to access only fragment $B$ be $q_i^B$. This means that $q_i^A$ and $q_i^B$ are not interfered each other before merging fragment $A$ and $B$. After merging them, $q_i^A$ and $q_i^B$ are interfered each other because both queries commonly access the merged fragment $C$. Hence, the merged fragment $C$ will create new interfered accesses.

Thus, in each step during constructing a PT we select two nodes (fragments) which can maximize the merging profit defined below when they are merged into a node (fragment).

**Merging profit:** $c \cdot d_{DF} - n_{IA}$

$d_{DF}$: the decreased $\#DF$

$n_{IA}$: the newly created $\#IA$

$c$: a proportional constant between $\#DF$ and $\#IA$

To select two nodes of $n$ nodes which can maximize the merging profit, $\binom{n}{2} = \frac{n(n-1)}{2}$ pairs should be examined. For example, because there are six nodes in Step 0 of Fig. 2, we examine the merging profits of $\frac{6(6-1)}{2} = 15$ pairs and merge one pair with the maximum merging profit among them, which generates the nodes in Step 1 of Fig. 2. Let us compute the merging profit of the pair of node $a_2$ and $a_3$ in Step 0. Data queries to access attribute $a_2$ and $a_3$ are $Q_{a_2} = \{q_1, q_2, q_4\}$ and $Q_{a_3} = \{q_1, q_4\}$, respectively, from the table of Fig. 2. When attribute $a_2$ and $a_3$ are merged, the value of $d_{DF}$ in the merging profit is $35 (=15+20)$ due to query $q_1$ and $q_4$. Before merging attribute $a_2$ and $a_3$, both $q_1$ and $q_4$ should access two different fragments independently. Since merging the two attributes makes query $q_1$ and $q_4$ access only one fragment, the access frequency of different fragments can be saved by the sum of the frequency of query $q_1$ and $q_4$. On the other hand, the value of $n_{IA}$ in the merging profit is 0 because there are no additionally interfered queries after merging. After all, the merging profit of attribute $a_2$ and $a_3$ is $c \cdot 35 - 0 = 35c$. Next, consider the pair of attribute $a_4$ and $a_5$ in Step 0 of Fig. 2. Attribute $a_4$ and $a_5$ are accessed by $Q_{a_4} = \{q_1, q_3\}$ and $Q_{a_5} = \{q_2, q_3\}$, respectively. In case of merging attribute $a_4$ and $a_5$, the value of $d_{DF}$ in the merging profit becomes 25 (i.e., the access frequency of query $q_3$) because the merging makes query $q_3$ only access one newly generated fragment $(a_4, a_5)$ instead of two fragments $a_4$ and $a_5$. And, the value of $n_{IA}$ is 25 which is equal to the sum of the frequency of query $q_1$ and $q_2$. As for attribute $a_4$ and $a_5$, query $q_1$ and $q_2$ are not interfered before merging the two attributes. However, because query $q_1$ and $q_2$ access the same fragment after merging, the two queries are newly interfered with each other. Therefore, the merging profit of attribute $a_4$ and $a_5$ in Step 0 of Fig. 2 is $25 \cdot c - 25$. When we assume that the proportional constant $c$ is 2, the merging profit of attribute $a_2$ and $a_3$ is maximum in Step 0. Because there are six nodes in Step 0 of Fig. 2, we should examine the merging profits of $\binom{6(6-1)}{2} = 15$ pairs and then merge one pair with the maximum merging profit among them. As in Table 1, $(a_2, a_3)$ and $(a_3, a_4)$ have the same maximum merging profit. In the example constructing a partition tree of Fig. 3, $(a_2, a_3)$ is first selected and merged to form Step 1, with expecting that $(a_5, a_6)$ will be merged in Step 1 to form Step 2. As a result, five nodes (fragments) in Step 1 of Fig. 2 are constructed by merging attribute $a_2$ and $a_3$ in Step 0.

We now propose an efficient scheme with which we can easily compute the merging profit and construct a PT. Each node (i.e., data fragment) $n_k$ in a PT has the data structure $Q_{n_k} = \{q_i, …, q_j\}$, which means that all queries in set $Q_{n_k}$ access the node. That is, node $a_2$ in Step 0 of Fig. 3 is accessed by query $q_1$, $q_2$, and $q_4$, and

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Merging profits in Step 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>10</td>
</tr>
<tr>
<td>$a_2$</td>
<td>35</td>
</tr>
<tr>
<td>$a_3$</td>
<td>-7</td>
</tr>
<tr>
<td>$a_4$</td>
<td>13</td>
</tr>
<tr>
<td>$a_5$</td>
<td>35</td>
</tr>
</tbody>
</table>
node \((a_4, a_5, a_6)\) in Step 3 is accessed by query \(q_1, q_2, q_3\). Let two nodes \(n_1\) and \(n_2\) have the data structure \(Q_{n_1} = \{q_i, \ldots, q_j\}\) and \(Q_{n_2} = \{q_s, \ldots, q_t\}\), respectively. In general, we can depict the relationship between \(Q_{n_1}\) and \(Q_{n_2}\) as the Venn diagram of Fig. 4. The Venn diagram is composed of three disjoint parts \(P_1, P_2,\) and \(P_3\), each of which can be an empty set (i.e., \(\emptyset\)). Because the queries within \(P_2\) access both node \(n_1\) and \(n_2\), the queries have only to access one newly generated node if the two nodes are merged. Thus, when we compute the merging profit of node \(n_1\) and \(n_2\), the value of \(d_{DF}\) can be calculated as follows:

\[
d_{DF} = \sum_{q_i \in (Q_{n_1} \cap Q_{n_2})} f_i
\]

In other words, \(d_{DF}\) is the sum of the frequency of data queries belonging to \(Q_{n_1} \cap Q_{n_2}\). On the other hand, data queries within \(P_1\) or \(P_3\) of Fig. 4 are not interfered with each other before merging node \(n_1\) and \(n_2\). However, because merging the two nodes makes the queries access the same node merged, the queries are newly interfered with each other after merging. Hence, the value of \(n_{JA}\) in the merging profit of node \(n_1\) and \(n_2\) can be computed as follows:

\[
n_{JA} = \begin{cases} 
\sum_{q_i \in (P_1 \cup P_3)} f_i & \text{if both } P_1 \neq \emptyset \text{ and } P_3 \neq \emptyset \\
0 & \text{otherwise}
\end{cases}
\]

Here, \(P_1\) and \(P_3\) can be denoted by \(Q_{n_1} - Q_{n_2}\) and \(Q_{n_2} - Q_{n_1}\), respectively. So far, we discussed how to simply compute the merging profit. If the merging profit of node \(n_1\) and \(n_2\) is known to be maximum, a new node \((n_1, n_2)\) with data structure \(Q_{(n_1, n_2)} = Q_{n_1} \cup Q_{n_2}\) is generated after merging the two nodes.

Using the proposed scheme to easily compute the merging profit, we can construct a PT as in Fig. 3 by repeatedly applying the process to select and merge two nodes with the maximum merging profit for each step of the PT.

### 3.2. Output data fragments

We can now decide the output data fragments of vertical partitioning in a PT according to the requirement of best-fit vertical partitioning or \(n\)-way vertical partitioning. Note that our PT can uniquely generate all kinds of vertical partitioning from a set of single-attribute data fragments to a big data fragment composed of all data attributes. For example, the PT in Fig. 3 generates six different fragment sets from Step 0 to Step 5, each of which is unique. Hence, we can easily support the \(n\)-way vertical partitioning problem, i.e., generating the specific number of fragments required. We have only to select a proper step in a PT which has the same...
number of nodes as the number of fragments required by the user.

On the other hand, for support of best-fit vertical partitioning, we need a method to identify which is an optimal partitioning among various kinds of vertical partitioning in a PT. In this paper, we discussed the cost model of vertical partitioning and its objective function \( \min[c \cdot \text{avg}(\#DF) + \text{avg}(\#IA)] \) that is a basis to determine an optimal partitioning. Therefore, best-fit vertical partitioning is to generate a collection of nodes (i.e., data fragments) within a step of a PT that meets the objective function. To compute the cost of \( [c \cdot \text{avg}(\#DF) + \text{avg}(\#IA)] \) of each step in a PT, we can use the data structure kept by each node. Consider a step in a PT where there are \( m \) nodes \( n_j \) (\( j = 1, \ldots, m \)) and each node \( n_j \) has the data structure \( Q_{nj} \). Let \( Q_{n} \) be the union of \( Q_{nj} \) (\( j = 1, \ldots, m \)), i.e., \( Q_{n} = \bigcup_{j=1}^{m} Q_{nj} \) while admitting duplicated elements. And, let \( k_i \) be the number of query \( q_i \) in the set \( Q_{n} \) and \( \text{Cardi}(Q_{n}) \) be the number of distinct queries in \( Q_{n} \). As an example, \( Q_{n} \) for Step 3 in Fig. 3 is \( Q_{n} = \{ q_{1}, q_{1}, q_{2}, q_{2}, q_{2}, q_{4}, q_{4} \} \). And, \( k_1, k_2, k_3, \) and \( k_4 \) are 2, 3, 1, and 1, respectively, and \( \text{Cardi}(Q_{n}) \) is 4. Using these notations, we can compute the value of \( \text{avg}(\#DF) \) as follows:

\[
\text{avg}(\#DF) = \frac{\sum_{i=1}^{m} [k_i - 1] \cdot f_i}{\text{Cardi}(Q_{n})}
\]

Here, \( f_i \) is the frequency of query \( q_i \). Hence, the value of \( \text{avg}(\#DF) \) for Step 3 in Fig. 3 is \( \frac{1(15) + 2(10) + 0(25) + 0(20)}{4} = \frac{35}{4} \).

On the other hand, when the number of nodes within Step \( S \) is \( n(S) \), the value of \( \text{avg}(\#IA) \) is computed by

\[
\text{avg}(\#IA) = \frac{\sum_{j=1}^{m} \#IA_{nj}}{n(S)}
\]

where \( \#IA_{nj} \) is the sum of the frequency of interfered accesses in node \( n_j \) as in the following.

\[
\#IA_{nj} = \begin{cases} \sum_{q_i \in Q_{nj}} f_i, & \text{if Cardi}(Q_{nj}) > 1 \\ 0, & \text{otherwise} \end{cases}
\]

For example, the value of \( \text{avg}(\#IA) \) of Step 3 in Fig. 3 is \( \frac{1(15) + 2(10) + 0(25) + 0(20)}{4} = \frac{35}{4} \). When we assume the proportional constant \( c \) is 2, best-fit vertical partitioning satisfying \( \min[c \cdot \text{avg}(\#DF) + \text{avg}(\#IA)] \) in Fig. 3 is the vertical partitioning of Step 3 that is composed of three fragments \( (a_1), (a_2, a_3), \) and \( (a_4, a_5, a_6) \).

Up to now, we have proposed our vertical partitioning method AVP. The AVP can adaptively support \( n \)-way vertical partitioning as well as best-fit vertical partitioning under the single vertical partitioning algorithm explained in the above. The overall algorithm of the AVP is described in Fig. 5.

**Theorem 3.1.** AVP generates a partition tree with maximum merging profit.

**Proof.** The proposed vertical partitioning method, AVP, is a greedy algorithm, because at each step during constructing a partition tree it merges two nodes with maximum merging profit. Note that AVP can be easily transformed into Kruskal’s algorithm that is also a greedy algorithm to find out a minimum spanning tree for every connected undirected graph. In Kruskal’s algorithm, an edge with minimum cost is determined at each step, thereby merging the two nodes. This operation is identical to that AVP merges two nodes with maximum merging profit. Here, the objective function of AVP, maximum merging profit, can be simply transformed into minimum cost of Kruskal’s algorithm.

```c
/* Assume that a data schema R has n data attributes. */
1. Initially, make n number of single-attribute fragments.

/* Construct a Partition Tree. */
2. WHILE (i = 1, · · · , n − 1) {
    2.1 Select a pair of two nodes with the maximum merging profit.
    2.2 Merge the pair and construct a new step in the PT.
}
/* In this point, a Partition Tree is constructed. */

/* Select a output partitioning. */
3. IF (best-fit vertical partitioning is required) {
    3.1 Select a step (i.e., a vertical partitioning) minimizing \( c \cdot \text{avg}(\#DF) + \text{avg}(\#IA)) \)
}
4. ELSE IF (n-way vertical partitioning is required) {
    4.1 Select a step with n number of nodes in the PT.
}

Fig. 5. AVP algorithm.
by making it be negative. The only difference between AVP and Kruskal’s algorithm is that the edge weight (i.e., cost) is statically fixed in Kruskal’s algorithm while AVP recomputes the edge weight (i.e., merging profit between two nodes) for each step. Even though we assume that Kruskal’s algorithm recomputes the edge weight for each step, its overall behavior is never affected. After all, we can transform AVP into Kruskal’s algorithm. On the other hand, it is already known that Kruskal’s algorithm generates a minimum cost spanning tree for every connected undirected graph (Horowitz and Sahni, 1978). Hence, the theorem follows. □

4. Experiments

In this section, we show several experimental results to evaluate the efficiency of the AVP method. Even though there are some up-to-date research such as Cornell and Yu (1990) and Chu and Leong (1993), we selected the binary vertical partitioning (BVP) method of Navathe et al. (1984) to compare with our AVP method because of the following two main reasons. First, the vertical partitioning approaches proposed in Cornell and Yu (1990) and Chu and Leong (1993) are limited to a single site environment. They considered that the response time of a query is strongly affected by the amount of data accessed from secondary storage (disk). Hence, their objective functions are to minimize the number of disk accesses. Nowadays, the cost to perform queries in distributed systems are, however, dominated by the remote network communication as well as local disk accesses. Therefore, the approaches proposed in Cornell and Yu (1990) and Chu and Leong (1993) cannot easily be extended to distributed systems. Second, as far as we are aware of, only Navathe et al. (1984) mentioned the n-way vertical partitioning problem and proposed a method to partially support it. Navathe et al. (1984) proposed the binary vertical partitioning (BVP) method for the best-fit vertical partitioning problem and supported the n-way vertical partitioning problem by the repetitive use of BVP.

In the experiments, the cost model proposed in this paper will be used as a basis to compare our AVP method with BVP of Navathe et al. (1984). Note that we need to decide the proportional constant c in the cost model to compute the cost of any vertical partitioning. We mentioned the effect of value c in the cost model of Section 2.2. Because the constant c is generally determined according to the target applications or system environments, we do not yet give any detail guidelines in deciding the value c. But, we assume that the value of c is 2.5 in the experiments. When the experiments with this value are performed, we found out that the results of AVP and BVP can be best comparable with each other. On the contrary, when the value of c is 1, it is very difficult to compare the two methods’ characteristics because their experimental results are too much different. The reason is that the BVP method is basically developed without considering the distributed system environment that are somewhat related with the cost factor #IA in the cost model. In addition, we empirically find out that #IA has no effect on vertical partitioning when c is greater than 5 in many cases.

For the first experiment, consider the information of attribute access patterns and query frequency in Fig. 6, which was used as an example in Navathe et al. (1984). That is, query \( q_1 \) accesses data attribute \( a_1, a_3, \) and \( a_7 \) with frequency 25. Table 2 shows the results of vertical partitioning by BVP and our AVP. Both methods for this example generate three fragments (i.e., \( R_1(a_1, a_5, a_7), R_2(a_2, a_3, a_6, a_9), \) and \( R_3(a_4, a_6, a_{10}) \)) as best-fit vertical partitioning. For the n-way vertical partitioning problem, our AVP can support it while BVP cannot fully support it as you can observe in the table. This is an critical limitation of the BVP method.

We use the example of Fig. 7 as the second experiment which was also used in Navathe et al. (1984). For best-fit vertical partitioning, both BVP and our AVP identically generate four fragments (i.e., \( R_1(a_1, a_4, a_5, a_6, a_8), R_2(a_2, a_9, a_{12}, a_{13}, a_{14}), R_3(a_3, a_7, a_{10}, a_{11}, a_{17}, a_{18}), \) and \( R_4(a_{15}, a_{16}, a_{19}, a_{20}) \)) as in Table 3. However, we can here notice two important differences between both algorithms for n-way vertical partitioning. One is that BVP does not fully support n-way vertical partitioning as in the first example. The other is that our AVP can always give one definitive solution for the n-way partitioning problem while BVP does not do it. For the requirement of 3-way partitioning, BVP provides two alternatives as in Table 3 and what is worse does not give any guidelines to select one of them. The results of 3-way partitioning are compared in Fig. 8 according to the cost model mentioned in Section 2.2. It is noticeable that our AVP method can always provide one definitive solution that is optimal. On the other hand, most previous research have only focused on minimizing the response time of query processing, which is related to the perfor-

![Fig. 6. The first example with 10 attributes and 8 queries.](image-url)
The objective of vertical partitioning in distributed systems is to improve system throughput as well as the performance of query execution for queries running on parts of data attributes. Vertical partitioning can be proceeded into two directions; one is to generate an overall optimal partitioning that minimizes the processing cost of data queries running on the fragments, which is called best-fit vertical partitioning in this paper. The other is to generate a specific number of fragments required by the user, which is generally called n-way vertical partitioning. Most previous approaches have only focused on the best-fit vertical partitioning problem. Although some research such as Navathe et al. (1984) discussed n-way vertical partitioning, their solutions can provide n-way vertical partitioning only in certain cases. In addition, because most previous approaches have considered the vertical partitioning problem under a single site environment, there are some limitations to extend them to the distributed system environment.

5. Summary

The objective of vertical partitioning in distributed systems is to improve system throughput as well as the performance of query execution for queries running on parts of data attributes. Vertical partitioning can be proceeded into two directions; one is to generate an overall optimal partitioning that minimizes the processing cost of data queries running on the fragments, which is called best-fit vertical partitioning in this paper. The other is to generate a specific number of fragments required by the user, which is generally called n-way vertical partitioning. Most previous approaches have only focused on the best-fit vertical partitioning problem. Although some research such as Navathe et al. (1984) discussed n-way vertical partitioning, their solutions can provide n-way vertical partitioning only in certain cases. In addition, because most previous approaches have considered the vertical partitioning problem under a single site environment, there are some limitations to extend them to the distributed system environment.
### Table 3

<table>
<thead>
<tr>
<th># of Fragments</th>
<th>Algorithm</th>
<th>Adaptive vertical partitioning (AVP)</th>
<th>Binary vertical partitioning (BVP)</th>
<th>Choice 1</th>
<th>Choice 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Fragment</td>
<td>R(a)</td>
<td>R(a)</td>
<td>R(a)</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>2 Fragments</td>
<td>R(a,b)</td>
<td>R(a,b)</td>
<td>R(a,b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Fragments</td>
<td>R(a,b,c)</td>
<td>R(a,b,c)</td>
<td>R(a,b,c)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Fragments</td>
<td>R(a,b,c,d)</td>
<td>R(a,b,c,d)</td>
<td>R(a,b,c,d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Fragments</td>
<td>R(a,b,c,d,e)</td>
<td>R(a,b,c,d,e)</td>
<td>R(a,b,c,d,e)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-20 Fragments</td>
<td>R(a,b,...)</td>
<td>R(a,b,...)</td>
<td>R(a,b,...)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this paper, we have first defined the cost model which can reflect the objective of vertical partitioning in distributed systems. And then, we have proposed an efficient and adaptable vertical partitioning method called AVP. The AVP method can support not only best-fit vertical partitioning but also \( n \)-way vertical partitioning.

### References


Jin Hyun Son is a professor of the Department of Computer Science and Engineering at Hanyang University, Ansan, Korea. He received his B.S. degree in Computer Science from Sogang University, Seoul, Korea, in 1996, and his M.S. and Ph.D. degrees in Computer Science from KAIST in 1998 and 2001, respectively. His research interests include database systems, distributed processing, object-oriented systems and e-Business.

Myoung Ho Kim is a professor of the Division of Computer Science at Korea Advanced Institute of Science and Technology (KAIST), Daejon, Korea, from 1989. He received his B.S. and M.S. degrees in Computer Engineering from Seoul National University, Seoul, Korea, in 1983 and 1984, respectively, and his Ph.D. degree in Computer Science from Michigan State University, East Lansing, MI, in 1989. His research interests include database systems, OLAP, mobile computing, transaction management, information retrieval, workflow and distributed processing. He is a member of the ACM and IEEE Computer Society.