

Cost-Based Fragmentation for Distributed Complex Value Databases

Hui Ma and Markus Kirchberg

Information Science Research Centre, Massey University,
Private Bag 11 222, Palmerston North, New Zealand
{h.ma,m.kirchberg}@massey.ac.nz

Abstract. The major purpose of the design of distributed databases is to improve system performance and to increase system reliability. Fragmentation and allocation play important roles in the development of a cost-efficient system. This paper addresses the problem of fragmentation in the context of complex value databases, which cover the common aspects of object-oriented databases, object-relational databases and XML. In this paper, we present a cost-based approach for horizontal and vertical fragmentation. Our approach is based on a cost model that takes the structure of complex value databases into account.

1 Introduction

Fragmentation and allocation are the main techniques used in the design of distributed databases. The research problems for fragmentation and allocation have been investigated in the context of the relational model and object-oriented model. With the current popularity of web information systems that often support web-based database application, including object-oriented databases, object-relational databases and databases based on the eXtensible Markup Language (XML), it is desired to have efficient and effective database design techniques that take into account the common aspects of these complex data models.

In the literature, fragmentation and allocation have been studied since the 1970s. They are often dealt with independently disregarding the fact that both design procedures rely on similar input information to achieve the same objectives, i.e. improve system performance and increase system reliability. Horizontal fragmentation is either performed with a set of minterm predicates [3, 8, 21] or with a set of predicates grouped according predicate affinities [1, 4, 22]. Horizontal fragmentation with minterm predicates often results in a big number of fragments, which will later be allocated to a small number of network nodes. Thus, fragment recombination is needed to match the required number of horizontal fragments, which is less or equal to the number of network nodes. Affinity-based horizontal fragmentation approaches cannot guarantee an optimal system performance as data locality information is lost while computing predicate affinities. In fact, neither of two approaches include local data availability as an objective of fragmentation.

For vertical fragmentation, algorithms proposed in the literature are either cost-driven or affinity-based. In the context of the relational model, cost-driven approaches are based on cost models, which measure the total number of disk accesses to the database [5, 7]. Affinity-based approaches group attributes according to attribute affinities [11, 17, 19, 20, 21]. Navathe *et al* [18] propose mixed fragmentation, which also uses affinities as the main parameters for both horizontal and vertical fragmentation. In the context of object-oriented data models, fragmentation is mainly affinity-based and different affinities, e.g. method affinities, attribute affinities or instance variable affinities [2, 9, 12], are used as parameters. Attribute affinities only reflect the togetherness of attributes accessed by applications. Vertical fragmentation based on affinities may reduce the number of disk accesses, but has never been proven to increase system performance. For distributed databases, transportation costs dominate the total query costs. This means that improving local data availability, which in turn reduces data transportation requirements, plays an important role towards improving system performance.

It is our aim to overcome the shortcomings of exiting approaches for both horizontal and vertical fragmentation. In this paper, we discuss both types of fragmentation in the context of complex value databases and present corresponding cost-based approaches. Thus, we extend earlier work as presented in [14, 16]. In short, we will incorporate all query information and also including site information by using a simplified cost model for fragmentation. This way, we can perform fragmentation and fragment allocation simultaneously. Properties of our approach include a low computational complexity and increased system performance.

The remainder of this paper is organised as follows: In Section 1, we briefly introduce a complex value data model. In Section 2, we define fragmentation techniques for the complex value data model. In Section 3, we present a cost model for complex value databases. In Section 4, we propose a cost-based design approach for horizontal and vertical fragmentation. We conclude with a short summary in Section 5.

1.1 Complex Value Databases

We define complex values on the basis of a type system. Using abstract syntax, types can be defined by

$$t = b \mid \mathbb{1} \mid (a_1 : t_1, \dots, a_n : t_n) \mid a : \{t\} \mid a : [t] \quad (1)$$

with pairwise different labels a_1, \dots, a_n . Here b represents a not further specified collection of base types, which may include *BOOL*, *CARD*, *INTEGER*, *DECIMAL*, *DATE*, etc. Furthermore, (\cdot) is a record type constructor, $\{\cdot\}$ a set type constructor, $[\cdot]$ a list type constructor and $\mathbb{1}$ is a trivial type.

Each type t defines a set of values, its *domain* $dom(t)$, as follows:

- The domain $dom(b_i)$ of a base type b_i is some not further specified set V_i , e.g. $dom(BOOL) = \{\mathbf{T}, \mathbf{F}\}$, $dom(CARD) = \{0, 1, 2, \dots\}$, etc.
- We have $dom(\mathbb{1}) = \{\top\}$.

- We have $\text{dom}((a_1 : t_1, \dots, a_n : t_n)) = \{(a_1 : v_1, \dots, a_n : v_n) \mid v_i \in \text{dom}(t_i)\}$ for record types.
- For set types we have $\text{dom}(a : \{t\}) = \{a : \{v_1, \dots, v_k\} \mid v_i \in \text{dom}(t)\}$.
- For list types we have $\text{dom}(a : [t]) = \{a : [v_1, \dots, v_k] \mid v_i \in \text{dom}(t)\}$, i.e. elements may appear more than once and are ordered.

In the following, we use the term *atomic attribute* to refer to an attribute with a base type as domain, and the term *complex attribute* to denote an attribute with a domain of a record type, set type or list type.

On the basis of this type system, we can define database schemata, which are sets of database types. A *database type of level k* has a name E and consists of a set $\text{comp}(E) = \{r_1 : E_1, \dots, r_n : E_n\}$ of components with pairwise different role names r_i and database types E_i on levels lower than k with at least one database type of level exactly $k - 1$, a set $\text{attr}(E) = \{a_1, \dots, a_m\}$ of attributes, each associated with a data type $\text{dom}(a_i)$ as its domain, and a key $\text{id}(E) \subseteq \text{comp}(E) \cup \text{attr}(E)$. We shall write $E = (\text{comp}(E), \text{attr}(E), \text{id}(E))$. A *database schema* is a finite set \mathcal{S} of database types such that for all $E \in \mathcal{S}$ and all $r_i : E_i \in \text{comp}(E)$ we also have $E_i \in \mathcal{S}$. That is, schemata are closed under component references.

Given a database schema \mathcal{S} , we associate two types $t(E)$ and $k(E)$, called *representation type* and *key type*, respectively, with each $E = (\{r_1 : E_1, \dots, r_n : E_n\}, \{a_1, \dots, a_k\}, \{r_{i_1} : E_{i_1}, \dots, r_{i_m} : E_{i_m}, a_{j_1}, \dots, a_{j_\ell}\}) \in \mathcal{S}$:

- The representation type of E is the tuple type $t(E) = (r_1 : k(E_1), \dots, r_n : k(E_n), a_1 : \text{dom}(a_1), \dots, a_k : \text{dom}(a_k))$.
- The key type of E is the tuple type $k(E) = (r_{i_1} : k(E_{i_1}), \dots, r_{i_m} : k(E_{i_m}), a_{j_1} : \text{dom}(a_{j_1}), \dots, a_{j_\ell} : \text{dom}(a_{j_\ell}))$.

Finally, a *database db* over a schema \mathcal{S} is an \mathcal{S} -indexed family $\{db(E)\}_{E \in \mathcal{S}}$ such that each $db(E)$ is a finite set of values of type $t(E)$ satisfying two conditions, with one as that whenever $t_1, t_2 \in db(E)$ coincide on their projection to $\text{id}(E)$, they are already equal; while the other one is that for each $t \in db(E)$ and each $r_i : E_i \in \text{comp}(E)$ there is some $t_i \in db(E_i)$ such that the projection of t onto r_i is t_i .

Example 1. The following is the complex value database schema for a simple university application:

```
DEPARTMENT = ( $\emptyset$ , {dname, homepage, contacts}, {dname})
LECTURER = ({in: DEPARTMENT}, {id, name, email, phone, homepage}, {id})
PAPER = ( $\emptyset$ , {no, ptitle, level, points}, {no})
LECTURE = ({paper: PAPER}, {semester, schedule}, {paper: PAPER, semester})
```

with $\text{dom}(\text{schedule}) = \{\text{slot} : (\text{time: } \textit{TIME}, \text{day: } \textit{STRING}, \text{room: } \textit{STRING})\}$ in the database type LECTURE. All other domains have been omitted.

The corresponding representation types for the university database schema are as follows:

$$t(\text{DEPARTMENT}) = (\text{dname: } \textit{STRING}, \text{homepage: } \textit{URI}, \text{contacts: } \{\text{contact: } (\text{location: } \textit{STRING}, \text{phone: } \textit{CARD})\})$$

$t(\text{LECTURER}) = (\text{in: (dname: } \textit{STRING}), \text{id: } \textit{STRING}, \text{name: (fname: } \textit{STRING}, \text{lname: } \textit{STRING}, \text{titles: \{title: } \textit{STRING}\}}, \text{email: } \textit{EMAIL}, \text{phone:(areacode: } \textit{CARD}, \text{number: } \textit{CARD}) \text{homepage: } \textit{URI})$
 $t(\text{PAPER}) = (\text{no: } \textit{CARD}, \text{ptitle: } \textit{STRING}, \text{level: } \textit{CARD}, \text{points: } \textit{CARD})$
 $t(\text{LECTURE}) = (\text{paper: (no: } \textit{CARD}), \text{semester: } \textit{STRING}, \text{schedule: \{slot: (time: } \textit{TIME}, \text{day: } \textit{STRING}, \text{room: } \textit{STRING}\}})$

1.2 A Query Algebra and Heuristic Query Optimisation

For complex value databases, to describe fragmentation and its effect on query optimisation, we need a query algebra. It is sufficient to take the recursive algebra for nested relations discussed in [6], because it allows to access data values at any level of complex values without any special navigational operation or flattening the complex values. Also, most of the query optimisation techniques developed for the relational algebra can be applied to queries expressed in the recursive algebra.

In the recursive algebra, the selection operation $\sigma_{\text{path}_i=c}(db(E))$ allows to define selection conditions on an attribute at any level using a path expression. The project operation $\pi_{\text{path}_i}(db(E))$ permits projections on attributes at all levels without restructuring. The join operation $db(E_1) \bowtie_{\text{path}_i} db(E_2)$ allows two instances to be joined on any attribute level defined by the path path_i . path_i is a list of attribute names starting from a top attribute name or reference name of a representation type, e.g. a_i , $a_i.a_{ij}$ or $r_i.a_i$.

2 Schema Fragmentation

Let us now introduce operations for fragmentation. Similar to the relational approach to horizontal fragmentation, we utilise the fact that each database type E defines a set $db(E)$ in a database db . Thus it can be partitioned into disjoint subsets.

2.1 Horizontal Fragmentation

As in any database db , the database type E is associated with a finite set $db(E)$. We obtain an easy generalisation of relational horizontal fragmentation. For this let E be some database type. Take boolean-valued functions φ_i ($i = 1, \dots, n$) such that for each database db we obtain

$$db(F_i) = \bigcup_{i=1}^n \sigma_{\varphi_i}(db(E))$$

with disjoint sets $\sigma_{\varphi_i}(db(E))$. We then replace E in the schema by n new database types F_i , all with the same definition as E .

Example 2. Let us consider an instance $db(\text{DEPARTMENT})$ (as outlined in Table 1) of database type DEPARTMENT from Example 1. Horizontal fragmentation with two predicates $\varphi_1 \equiv \text{contacts.contact.location} = \text{'Turitea'}$ and $\varphi_2 \equiv \text{contacts.contact.location} \neq \text{'Turitea'}$ results in two fragments $db(\text{TURITEA_DEPARTMENT})$ and $db(\text{NO_TURITEA_DEPARTMENT})$ as outlined in Table 1.

Table 1. HORIZONTAL FRAGMENTATION OF $db(\text{DEPARTMENT})$

$db(\text{DEPARTMENT})$			
dname	homepage	contacts	
		location	phone
Information	is.massey.ac.nz	Turitea	063566199
		Wellington	045763112
Computer	cs.massey.ac.nz	Turitea	063563188
		Albany	098132699
Education	ed.massey.ac.nz	Napier	068355202
		Albany	094437900

$db(\text{TURITEA_DEPARTMENT})$			
dname	homepage	contacts	
		location	phone
Information	is.massey.ac.nz	Turitea	063566199
Computer	cs.massey.ac.nz	Turitea	063563188

$db(\text{NO_TURITEA_DEPARTMENT})$			
dname	homepage	contacts	
		location	phone
Information	is.massey.ac.nz	Wellington	045763112
Computer	cs.massey.ac.nz	Albany	098132699
Education	ed.massey.ac.nz	Napier	068355202
		Albany	094437900

2.2 Vertical Fragmentation

Taking a database type $E = (\{r_1 : E_1, \dots, r_n : E_n\}, \{a_1, \dots, a_k\}, \{r_{i_1} : E_{i_1}, \dots, r_{i_m} : E_{i_m}, a_{j_1}, \dots, a_{j_\ell}\})$, vertical fragmentation on E replaces E with a set of new types F_1, \dots, F_m with $F_j = (\{r_1^j : E_1^j, \dots, r_n^j : E_n^j\}, \{a_1^j, \dots, a_k^j\}, \{r_{i_1}^j : E_{i_1}^j, \dots, r_{i_m}^j : E_{i_m}^j, a_{j_1}^j, \dots, a_{j_\ell}^j\})$ such that:

- the components and attributes will be distributed:

$$\{E_1, \dots, E_n\} = \bigcup_{j=1}^m \{E_1^j, \dots, E_n^j\}, \{a_1, \dots, a_k\} = \bigcup_{j=1}^m \{a_1^j, \dots, a_k^j\},$$

- $db(E)$ is split into $db(F_1), \dots, db(F_m)$ such that $db(E)$ could be reconstructed by using the join operation on all the instances:

$$db(E) = db(F_1) \bowtie \dots \bowtie db(F_m).$$

Using the query algebra, vertical fragmentation could be written as $db(F_j) = \pi_{F_j}(db(E))$ for all $j \in \{1, \dots, m\}$. To meet the criteria of reconstructivity, it is

Table 2. An Instance of LECTURE

<i>db</i> (LECTURE)				
paper no	semester	schedule		
		time	day	room
157111	0702	10am	Monday	SSLB1
		1pm	Wednesday	SSLB5
157221	0701	9am	Tuesday	AH1
		10am	Friday	AH2
157331	0701	1pm	Tuesday	SSLB3
		3pm	Thursday	SSLB2

<i>db</i> (LECTURE_TIME)			
paper no	semester	schedule	
		1 time	day
157111	0702	1 10am	Monday
		2 1pm	Wednesday
157221	0701	1 9am	Tuesday
		2 10am	Friday
157331	0701	1 1pm	Tuesday
		2 3pm	Thursday

<i>db</i> (LECTURE_VENUE)		
paper no	semester	schedule
		1 room
157111	0702	1 SSLB1
		2 SSLB5
157221	0701	1 AH1
		2 AH2
157331	0701	1 SSLB3
		2 SSLB2

required that the key type $k(E)$ is part of all types F_j . In addition, when projection is applied on attributes of a tuple type within a set type, e.g. $\{(a_{i1}, \dots, a_{in})\}$, an index I should be inserted as an arrogant attribute in the set constructor before fragmentation. This index should later be attached to each of the vertical fragments to ensure the reconstructivity.

Example 3. Let us consider the database type LECTURE from Example 1 and alter the representation type t_{LECTURE} to $t_{\text{LECTURE}'}$ by attaching an index attribute as a subattribute of attribute ‘slot’. Assume that we are given two subtypes $t_{\text{LECTURE_TIME}}$ and $t_{\text{LECTURE_VENUE}}$, each containing a subset of the attributes from t_{LECTURE} :

$$t_{\text{LECTURE}'} = (\text{paper: (no: CARD), semester: STRING, schedule: \{slot: (I: CARD, time TIME, day: STRING, room: STRING)\}});$$

$$t_{\text{LECTURE_VENUE}} = (\text{paper: (no: CARD), semester: STRING, schedule: \{slot: (I: CARD, room: STRING)\}});$$

$$t_{\text{LECTURE_TIME}} = (\text{paper: (no: CARD), semester: STRING, schedule: \{slot: (I: CARD, time: TIME, day: STRING)\}}).$$

Accordingly, we get the two vertical fragments that result from project operations:

$$db(\text{LECTURE_TIME}) = \pi_{t_{\text{LECTURE_TIME}}}(db(\text{LECTURE})) \text{ and}$$

$$db(\text{LECTURE_VENUE}) = \pi_{t_{\text{LECTURE_VENUE}}}(db(\text{LECTURE})).$$

Analogously, the instances of type LECTURE and the resulting fragments are shown in Table 2.

3 A Cost Model

In order to measure system performance, we need a cost model to compute total query costs for the queries represented by query trees. In this section, we first present formulae for estimating the sizes of intermediate results for all intermediate nodes in the query tree. These sizes determine the costs for retrieving data for the next step, i.e. the operation associated with the predecessor in the query tree, and the costs for the transportation of data between nodes. Afterwards, we present a cost model for measuring system performance.

3.1 Size Estimation

In order to estimate the size of leaf nodes and intermediate nodes that are of complex values, we first look at types t , and estimate the size $s(t)$ of a value of type t , which depends on the context. Then, the size of an instance $db(E)$ is $n_E \cdot s(t(E))$, where n_E is the average number of elements in $db(E)$.

Let s_i be the average size of elements for a base type b_i . This can be used to determine the size $s(t)$ of an arbitrary type t , i.e. the average space needed for it on storage. We obtain:

$$s(t) = \begin{cases} s_i & \text{if } t = b_i \\ \sum_{i=1}^n s(t_i) & \text{if } t = (a : t_1, \dots, a_n : t_n) \\ r \cdot s(t') & \text{if } t = \{t'\} \text{ or } t = [t']. \end{cases}$$

In the last of these cases, r is the average number of elements in sets, $t = [t']$ or $t = \langle t' \rangle$, respectively, within a value of type t .

Then, for $E = (\{r_1 : E_1, \dots, r_n : E_n\}, \{a_1, \dots, a_k\}, id(E))$ we obtain:

$$s(t(E)) = \sum_{i=1}^n s(t(E_i)) + \sum_{j=1}^k s(a_j).$$

The calculation of sizes of database instances applies also to the intermediate results of all queries. However, we can restrict our attention to the nodes of selection and projection, as the other nodes in the query tree will not be affected by fragmentation and subsequent heuristic query optimisation [16].

- The size of a selection node σ_φ is $p \cdot s$, where s is the size of the successor node and p is the probability that a tuple in the successor will satisfy φ .
- The size of a projection node π_X is $(1 - c) \cdot s \cdot \frac{s(t_X)}{s(t)}$ where t is the representation type of a complex value database type associated with the successor node, and t_X is the representation type associated with the projection node.

For sizes of results for other algebra operations, refer to the work in [13].

3.2 Query Processing Costs

Taking the cost model in [15], we now analyse the query costs in the case of fragmentation. For the convenience of discussion, we briefly present the cost model first. The major objective is to base the fragmentation decision on the efficiency of the most frequent queries.

Fragmentation results in a set of fragments $\{F_1, \dots, F_n\}$ of average sizes s_1, \dots, s_n . If the network has a set of nodes $N = N_1, \dots, N_k$ we have to allocate these fragments to one of the nodes, which gives rise to a mapping $\lambda : \{1, \dots, n\} \rightarrow \{1, \dots, k\}$, which we call a *location assignment*. This decides the allocation of leaves of query trees, which are fragments. For each intermediate node v in each relevant query tree, we must also associate a node $\lambda(v)$. $\lambda(v)$ indicates the node in the network that the intermediate query result, which corresponds to v , will be stored at.

Given a location assignment λ , we can compute the total costs of query processing. Let the set of queries be $Q^m = \{Q_1, \dots, Q_m\}$, each of which is executed with a frequency f_j . The total costs of all the queries in Q^m are the sum of the costs of each query multiplied by its frequency. The cost of each query are composed of two parts, the storage costs and transportation costs. Storage costs measure the costs of retrieving the data back from secondary storage. Those costs depend on the size of the intermediate results and on the assigned locations, which determine the storage cost factors. The transportation costs provide a measure for data transmission between two nodes of the network. Such transportation costs depend on the sizes of the involved sets and on the assigned locations, which determine the transport cost factor between every pair of sites.

$$\begin{aligned} Costs_\lambda(Q^m) &= \sum_{j=1}^m (stor_\lambda(Q_j) + trans_\lambda(Q_j)) \cdot f_j \\ &= \sum_{j=1}^m \left(\sum_h s(h) \cdot d_{\lambda(h)} + \sum_h \sum_{h'} c_{\lambda(h')\lambda(h)} \cdot s(h') \right) \cdot f_j \end{aligned}$$

where h ranges over the nodes of the query tree for Q_j , $s(h)$ are the sizes of the involved sets, and d_i indicates the storage cost factor for node N_i ($i = 1, \dots, k$), h' runs over the predecessors of h in the query tree, and c_{ij} is the transportation cost factor for data transport from node N_i to node N_j ($i, j \in \{1, \dots, k\}$).

Because transportation costs dominate the total query costs, we can develop cost minimisation heuristics by considering query frequencies, transportation cost factors and the size of data that need to be transferred between network sites. As discussed in [16, 14, 15], the allocation of fragments to sites according to cost minimisation heuristics already determines the location assignment provided that an optimal location assignment is given prior to the fragmentation.

4 A Cost-Based Methodology for Fragmentation

In this section, we present a cost-based approach for horizontal and vertical fragmentation. In each of the following sections, we first define some terms to

facilitate the discussion of fragmentation. Then, we present algorithms for fragmentation based on the analysis on the cost model in Section 3. In the following, we assume a database instance $db(E)$ with a representation type $t(E)$ being accessed by a set of the most frequent queries $Q^m = \{Q_1, \dots, Q_j, \dots, Q_m\}$, with frequencies f_1, \dots, f_m , respectively.

4.1 A Cost-Based Approach for Horizontal Fragmentation

Given a list of sorted queries, which access the instance of a given database type, by decreasing frequency $\mathcal{Q} = [Q_1, \dots, Q_j, \dots, Q_m]$, we obtain a set of simple predicates needed for horizontal fragmentation using *Num_Simple_Predicates* [15]. Let $\Phi^m = \{\varphi_1, \dots, \varphi_m\}$ be the chosen set of simple predicates defined on a database type E . Then, the set of *normal predicates* $\mathcal{N}^m = \{\mathcal{N}_1, \dots, \mathcal{N}_n\}$ on relation schema E is the set of all satisfiable predicates of the form $\mathcal{N}_j \equiv \varphi_1^* \wedge \dots \wedge \varphi_m^*$, where φ_i^* is either φ_i or $\neg\varphi_i$. Normal predicates can be represented in the following form:

$$\mathcal{N}_j = \bigwedge_{i \in J} \varphi_i \wedge \bigwedge_{i \notin J} \neg\varphi_i.$$

with $J \subseteq \{1, \dots, m\}$ as a set of indices of a subset of all simple predicates. Let f_i be the frequency of predicate φ_i , $J_\theta = \{i | i \in J \wedge \varphi_i \text{ executed at site } \theta\}$ be a subset of indices of all simple predicates, executed at site N_θ . We define the following terms:

Definition 1. An *atomic horizontal fragment* F_j is a fragment that is defined by a normal predicate:

$$F_j = \sigma_{\mathcal{N}_j}(E).$$

Definition 2. The *request* of an atomic fragment at site θ is the sum of frequencies of predicates issued at site θ :

$$request_\theta(F_j) = \sum_{j=1, j \in J_\theta}^k f_j.$$

Definition 3. The *pay* of allocating an atomic horizontal fragment at a site θ is the costs of accessing the atomic horizontal fragment by all queries from sites other than θ :

$$pay_\theta(F_j) = \sum_{\theta'=1, \theta' \neq \theta}^k request_{\theta'}(F_j) \cdot c_{\theta\theta'}.$$

With the terms defined above, we now present an algorithm of horizontal fragmentation as shown in Table 3. The algorithm first finds the site that has the biggest value of *pay* of each atomic fragment and then allocates the atomic fragment to the site. A fragmentation schema and fragment allocation schema can be obtained simultaneously.

An evaluation has been conducted with satisfiable results. For detailed discussion of horizontal fragmentation refer to [15].

Table 3. Algorithm for Horizontal Fragmentation

Input:	$\Phi^y = \{\varphi_1, \dots, \varphi_y\}$ /* a set of simple predicates a set of network nodes $N = \{1, \dots, k\}$ with cost factors c_{ij}
Output:	Horizontal fragmentation and allocation schema $\{F_{H1}, \dots, F_{Hk}\}$
Method:	<pre> for each $\theta \in \{1, \dots, k\}$ $F_{H\theta} = \emptyset$ endfor define a set of normal predicates \mathcal{N}^y using Φ^y define a set of atomic horizontal fragments \mathcal{F}^y using \mathcal{N}^y for each atomic fragment $F_j \in \mathcal{F}^y$, $1 \leq i \leq 2^y$ do for each node $\theta \in \{1, \dots, k\}$ do calculate $request_\theta(F_j)$ calculate $pay_\theta(F_j)$ endfor choose w such that $pay_w(F_j) = \min(pay_1(F_j), \dots, pay_k(F_j))$ /* find the minimum value $\lambda(F_j) = \mathcal{N}_w$ /* allocate F_j to the site of the smallest pay define $F_{H\theta}$ with $F_{H\theta} = \bigcup \{F_j : \lambda(F_j) = N_\theta\}$ endfor </pre>

4.2 A Cost-Based Approach for Vertical Fragmentation

In this section, we adapt the cost-efficient vertical fragmentation approach from [10, 14] to complex value databases. We start with some terminology, continue to present a vertical fragmentation design methodology and, finally, illustrate it with an example.

If $db(E)$ is vertically fragmented into a set of fragments $\{F_{V1}, \dots, F_{Vu}, \dots, F_{Vk_i}\}$, each of the fragments will be allocated to one of the network nodes $N_1, \dots, N_\theta, \dots, N_k$. Note that the maximum number of fragments is k , i.e. $k_i \leq k$. Let $\lambda(Q_j)$ indicate the site issuing query Q_j , $atomic(E) = \{a_1, \dots, a_n\}$ indicate the set of atomic attributes of E , f_{ji} be the frequency of the query Q_j accessing a_i . Here, $f_{ji} = f_j$ if the attribute a_i is accessed by Q_j . Otherwise, $f_{ji} = 0$.

Definition 4. The *request* of an attribute at a site θ is the sum of frequencies of all queries at the site accessing the attribute:

$$request_\theta(a_i) = \sum_{j=1, \lambda(Q_j)=\theta}^m f_{ji}.$$

Definition 5. The *pay* of allocating an attribute a_i to a site θ measures the costs of accessing attribute a_i by all queries from sites θ' other than site θ :

$$pay_\theta(a_i) = \sum_{\theta'=1, \theta \neq \theta'}^k \sum_{j=1, \lambda(Q_j)=\theta'}^m f_{ji} \cdot c_{\theta\theta'}.$$

Table 4. Algorithm for Vertical Fragmentation

Input:	the <i>AUFM</i> of E $atomic(E) = \{a_1, \dots, a_n\}$ /* a set of atomic attribute $PATH(E) = \{path_i, \dots, path_n\}$ /* a set of path of all atomic attributes a set of network nodes $N = \{1, \dots, k\}$ with cost factors c_{ij}
Output:	vertical fragmentation and fragment allocation schema $\{F_{V1}, \dots, F_{Vk}\}$
Method:	<pre> for each $\theta \in \{1, \dots, k\}$ let $atomic(F_{V\theta}) = k(E)$ endfor for each attribute $a_i \in atomic(E), 1 \leq i \leq n$ do for each node $\theta \in \{1, \dots, k\}$ do calculate $request_\theta(a_i)$ endfor for each node $\theta \in \{1, \dots, k\}$ do calculate $pay_\theta(a_i)$ endfor choose w such that $pay_w(a_i) = \min_{\theta=1}^k pay_\theta(a_i)$ $atomic(F_{Vw}) = atomic(F_{Vw}) \cup \{a_i\}$ /* add a_i to F_{Vw} $PATH(F_{Vw}) = PATH(F_{Vw}) \cup \{path_i\}$ /* add $path_i$ to $PATH(F_{Vw})$ endfor for each $\theta \in \{1, \dots, k\}, F_{Vw} = \pi_{PATH(F_{Vw})}(db(E))$ endfor </pre>

In order to record query information, we use an *Attribute Usage Frequency Matrix* (AUFM), which records frequencies of queries, the set of atomic attributes accessed by the queries and the sites that issue the queries. Each row in the AUFM represents one query Q_j ; the head of each column contains the set of attributes of a given representation type $t(E)$, the site issuing the queries and the frequency of the queries. Here, we do not distinguish between references and attributes, but record them in the same matrix. The values on a column indicate the frequencies f_{ji} of the queries Q_j that use the corresponding atomic attribute a_i grouped by the site that issues the queries. Note that we treat any two queries issued at different sites as different queries, even if the queries themselves are the same. The AUFM is constructed according to optimised queries in order to record all the attribute requirements returned by queries as well as all the attributes used in some join predicates. If a query returns all the information of an attribute then all its sub-attributes are accessed by the query.

With the *AUFM* as an input, we now present a vertical fragmentation algorithm as described in Table 4. For each atomic attribute at each site, the algorithm first calculates the *request* and then calculates the *pay*. At last, all atomic attributes are clustered to the site that has the lowest value of the *pay*. Correspondingly, a set of path expressions for each vertical fragment are obtained. Vertical fragmentation is performed by using the sets of paths. A vertical fragmentation and an allocation schema are obtained simultaneously.

Using the algorithm in 4 we can always guarantee that the resulting vertical fragmentation schema meet the criteria of correctness rules. Disjointness and completeness are satisfied because all atomic attributes occur and only occur in one of the fragments. Reconstruction is guaranteed because all fragments are composed of key attributes. In addition, if an attribute with a domain of a type

Table 5. Attribute *Usage* Frequency Matrix

site	query	frequency	in		name			email	phone		homepage
			id	dname	fname	lname	titles		areacode	number	
length			20	20	20 · 8	20 · 8	2 · 15 · 8	30 · 8	10	20	50 · 8
1	Q_1	20	20	20	20	20	20	20	20	20	20
	Q_4	50	0	0	50	0	0	50	0	0	0
2	Q_2	30	0	0	0	30	30	0	0	0	30
	Q_5	70	0	0	0	0	70	0	70	0	0
3	Q_3	100	0	0	0	100	0	0	100	100	0

Table 6. Attribute *Request* Matrix

request	in		name			email	phone		homepage
	id	dname	fname	lname	titles		areacode	number	
$request_1(a_i)$	20	20	70	20	20	70	20	20	20
$request_2(a_i)$	0	0	0	30	100	0	70	0	30
$request_3(a_i)$	0	0	0	100	0	0	100	100	0

inside a set type is decomposed, an index is attached to the attribute before vertical fragmentation, which will then be attached to each of fragments.

Example 4. We now illustrate the algorithm using an example. Assume there are five queries that constitute the 20% most frequently executed queries, which access an instance of database type LECTURER from three different sites:

- Query 1 $\pi_{LECTURER}(\sigma_{name.titles \ni 'Professor'}(LECTURER))$ issued at site 1 with $f_1 = 20$,
- Query 2 $\pi_{titles, homepage}(LECTURER)$ issued at site 2 with $f_2 = 30$,
- Query 3 $\pi_{name.lname, phone}(LECTURER)$ issued at site 3 with $f_3 = 100$,
- Query 4 $\pi_{fname, email}(LECTURER)$ issued at site 1 with $f_4 = 50$, and
- Query 5 $\pi_{titles, areacode}(LECTURER)$ issued at site 2 with $f_5 = 70$.

In order to perform vertical fragmentation using the design procedure as introduced in Section 4.2, we first construct an Attribute *Usage* Frequency Matrix as shown in Table 5. Secondly, we compute the *request* for each attribute at each site, the results of which are shown in the Attribute *Request* Matrix in Table 6. Thirdly, assuming the values of transportation cost factors are: $c_{12} = c_{21} = 10$, $c_{13} = c_{31} = 25$, $c_{23} = c_{32} = 20$, we can now calculate the *pay* for each attribute at each site using the values of the *request* from Table 6. The results are shown in an Attribute *Request* Matrix in Table 6 and an Attribute *Pay* Matrix in Table 7.

Table 7. Attribute *Pay* Matrix

<i>pay</i>	in	id	name			email	phone		homepage
	dname		fname	lname	titles		areacode	number	
					title				
$pay_1(a_i)$	0	0	0	2800	1000	0	3200	2500	300
$pay_2(a_i)$	200	200	700	2200	200	700	2200	2200	200
$pay_3(a_i)$	500	500	1750	1100	2500	1750	1900	500	1100
site	1	1,2,3	1	3	2	1	3	3	2

Once atomic attributes are grouped and allocated to sites, we get a set of paths for each site to be used for vertical fragmentation:

- $db(F_{V1}) = \pi_{id, in.dname, name.fname, email}(db(LECTURE))$,
- $db(F_{V2}) = \pi_{id, name.titles.title, homepage}(db(LECTURE))$ and
- $db(F_{V3}) = \pi_{id, name.lname, phone}(db(LECTURE))$

We now look at how the system performance is changed due to the outlined fragmentation by using the cost model presented above. Assume that the average number of lecturers is 20 and the average number of titles for each lecturer is 2. With the average length of each attribute given in Table 5, we can compute the total query costs. Assume that distributed query processing and optimisation are supported, then, selection and projection should be processed first locally to reduce the size of data transported among different sites. In this case, the optimised allocation of $db(LECTURERS)$ is site 2, which leads to total query costs of 16,680,000 while the total query costs after the vertical fragmentation and allocation are 4,754,000, which is about one fourth of the costs before the fragmentation. This shows that vertical fragmentation can indeed improve system performance.

4.3 Discussion

In order to obtain optimised fragmentation and allocation schemata for complex value databases, a cost model should be involved to evaluate the system performance. However, due to the complexity of fragmentation and allocation it is practically impossible to achieve an optimal fragmentation and allocation schema by exhaustively comparing different fragmentation schemata and allocation schemata using the cost model. However, from the cost model above, we observe that the less the value of the pay of allocating an attribute or an atomic fragment to a site the less the total costs will be to access it [14]. This explains that the proposed cost-based fragmentation approach above can at least determine a semi-optimal vertical fragmentation schema.

5 Conclusion

In this paper, we presented a cost-driven approach for fragmentation in complex value databases. This approach takes into consideration the structure of complex

value databases. Furthermore, algorithms are presented for each of the fragmentation techniques used in distribution design to obtain fragmentation schemata, which can indeed improve the system performance.

A related problem left for future work is a design methodology to integrate the use horizontal and vertical fragmentation techniques in design distributed databases.

References

- [1] Bellatreche, L., Karlapalem, K., Simonet, A.: Algorithms and support for horizontal class partitioning in object-oriented databases. *Distributed and Parallel Databases* 8(2), 155–179 (2000)
- [2] Bellatreche, L., Simonet, A., Simonet, M.: Vertical fragmentation in distributed object database systems with complex attributes and methods. In: Wagner, H.T.R. (ed.) *Seventh International Workshop on Database and Expert Systems Applications, DEXA '96*, pp. 15–21. IEEE-CS Press, Los Alamitos (1996)
- [3] Ceri, S., Negri, M., Pelagatti, G.: Horizontal data partitioning in database design. In: *Proc. the ACM SIGMOD International Conference on Management of Data*, pp. 128–136. ACM Press, New York (1982)
- [4] Cheng, C.-H., Lee, W.-K., Wong, K.-F.: A genetic algorithm-based clustering approach for database partitioning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C* 32(3), 215–230 (2002)
- [5] Chu, P.-C.: A transaction oriented approach to attribute partitioning. *Information Systems* 17(4), 329–342 (1992)
- [6] Colby, L.S.: A recursive algebra and query optimization for nested relations. In: *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data, Portland, Oregon, United States*, pp. 273–283. ACM Press, New York, NY, USA (1989)
- [7] Cornell, D., Yu, P.: A vertical partitioning algorithm for relational databases. In: *International Conference on Data Engineering, Los Angeles, California*, pp. 30–35 (1987)
- [8] Ezeife, C.I., Barker, K.: A comprehensive approach to horizontal class fragmentation in a distributed object based system. *Distributed and Parallel Databases* 3(3), 247–272 (1995)
- [9] Ezeife, C.I., Barker, K.: Vertical fragmentation for advanced object models in a distributed object based system. In: *Proceedings of the 7th International Conference on Computing and Information*, pp. 613–632. IEEE Computer Society Press, Los Alamitos (1995)
- [10] Hartmann, S., Ma, H., Schewe, K.-D.: Cost-based vertical fragmentation for xml. In: *DBMAN 2007, Springer, Heidelberg* (to appear 2007)
- [11] Hoffer, J.A., Severance, D.G.: The use of cluster analysis in physical database design. In: *Proceedings of the First International Conference on Very Large Data Bases, Framingham, MA* (September 1975)
- [12] Karlapalem, K., Navathe, S.B., Morsi, M.M.A.: Issues in distribution design of object-oriented databases. In: *IWDOM*, pp. 148–164 (1992)
- [13] Ma, H.: Distribution design in object oriented databases. Master's thesis, Massey University (2003)

- [14] Ma, H., Schewe, K.-D., Kirchberg, M.: A heuristic approach to vertical fragmentation incorporating query information. In: Vasilecas, O., Eder, J., Caplinskas, A. (eds.) Proceedings of the 7th International Baltic Conference on Databases and Information Systems, pp. 69–76. IEEE Computer Society Press, Los Alamitos (2006)
- [15] Ma, H., Schewe, K.-D., Wang, Q.: A heuristic approach to cost-efficient fragmentation and allocation of complex value databases. In: Bailey, G.D.J. (ed.) Proceedings of the 17th Australian Database Conference, Hobart, Australia. CRPIT 49, pp. 119–128 (2006)
- [16] Ma, H., Schewe, K.-D., Wang, Q.: Distribution design for higher-order data models. *Data and Knowledge Engineering* (to appear 2007)
- [17] Muthuraj, J., Chakravarthy, S., Varadarajan, R., Navathe, S.B.: A formal approach to the vertical partitioning problem in distributed database design. In: Proceedings of the Second International Conference on Parallel and Distributed Information Systems, San Diego, CA, USA, January 1993, pp. 26–34 (1993)
- [18] Navathe, S., Karlapalem, K., Ra, M.: A mixed fragmentation methodology for initial distributed database design. *Journal of Computer and Software Engineering* 3(4) (1995)
- [19] Navathe, S.B., Ceri, S., Wiederhold, G., Dour, J.: Vertical partitioning algorithms for database design. *ACM TODS* 9(4), 680–710 (1984)
- [20] Navathe, S.B., Ra, M.: Vertical partitioning for database design: A graphical algorithm. *SIGMOD Record* 14(4), 440–450 (1989)
- [21] Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems. Alan Apt, New Jersey (1999)
- [22] Zhang, Y.: On horizontal fragmentation of distributed database design. In: Orłowska, M., Papazoglou, M. (eds.) *Advances in Database Research*, pp. 121–130. World Scientific Publishing, Singapore (1993)