Design and Evaluation of Data Allocation Algorithms for Distributed Multimedia Database Systems

Yu-Kwong Kwok, *Member, IEEE*, Kamalakar Karlapalem, *Member, IEEE*, Ishfaq Ahmad, *Member, IEEE*, and Ng Moon Pun

***Abstract*— A major cost in retrieving multimedia data from multiple sites is the cost incurred in transferring multimedia data objects (MDO's) from different sites to the site where the query is initiated. The objective of a data allocation algorithm is to locate the MDO's at different sites so as to minimize the total data transfer cost incurred in executing a given set of queries. There is a mutual dependency between data allocation and query execution strategies in that the optimal allocation of MDO's depends on the query execution strategy employed by a distributed multimedia system while the query execution strategy optimizes a query based on this allocation. In this paper, we fix the query execution strategy and develop a site-independent MDO dependency graph representation to model the dependencies among the MDO's accessed by a query. Given the MDO dependency graphs as well as the set of multimedia database sites, data transfer costs between the sites, the allocation limit on the number of MDO's that can be allocated at a site, and the query execution frequencies from the sites, an allocation scheme is generated. We formulate the data allocation problem as an optimization problem. We solve this problem with a number of techniques that broadly belong to three classes: max-flow min-cut, state-space search, and graph partitioning heuristics. The max-flow min-cut technique formulates the data allocation problem as a network-flow problem, and uses a hill-climbing approach to try to find the optimal solution. For the state-space search approach, the problem is solved using a best-first search algorithm. The graph partitioning approach uses two clustering heuristics, the agglomerative clustering and divisive clustering. We evaluate and compare these approaches, and assess their cost-performance trade-offs. All algorithms are also compared with optimal solutions obtained through exhaustive search. Conclusions are also made on the suitability of these approaches to different scenarios.**

***Index Terms*— Data allocation, distributed database systems, multimedia database systems, query processing, hill-climbing heuristics, optimal allocation, max-flow min-cut problem, network flow algorithm, clustering, best-first search algorithm.**

I. Introduction

A distributed multimedia database system [13], [19] is a database system loosely coupled with a multimedia data provider as shown in Fig. 1 (unlike the integrated and heterogeneous computing paradigm based distributed multimedia database systems [2]). Special-purpose distributed multimedia database system architectures [8], [18] have been proposed. The aim of these architectures is to support specific application domains (like, medical databases and news-on-demand), and thus they are tightly integrated to favor these applications. We consider a general-purpose loosely coupled architecture which can be built by using off-the-shelf products. In this architecture, multimedia data provider (MDP) enables users to retrieve multimedia data objects (MDO's) from different sites. A common multimedia user interface (CMUI) enables the users to specify queries accessing the distributed multimedia database system and presenting the result to the user. The synchronization for the presentation of the multimedia data is handled by the CMUI [15]. Whereas, the MDP identifies the relevant multimedia data for an user query and facilitates shipping of the multimedia data to the CMUI. The CMUI is a client process and the distributed database management system (DDBMS) and the MDP are server processes. There can be multiple CMUI processes, and multiple DDBMS and MDP processes. An user can use any one of the CMUI client processes to query the distributed multimedia database system. The query is decomposed into two parts by CMUI: one accessing the multiple DDBMS servers, and the other accessing multiple MDP servers. After which, the DDBMS and MDP servers ship the result data to the CMUI. Finally, the result is presented by the CMUI to the user.

A major component of multimedia query execution cost is the data transfer cost. That is, the total cost involved in moving the MDO's from the sites where they are located to the site where the query is issued. The MDO's are made of two kinds of data. The first is the single-media data that is managed by the DDBMS servers, such as relations (fragments), records, etc. The second is the multimedia data, such as audio, video, and image, managed by the MDP servers. These two types of data are managed by different specialized storage managers, and need to be retrieved for the user queries. As the allocation problem arises for both single-media data and multimedia data, we develop a common data allocation problem formulation and a solution methodology. Since the CMUI is responsible for synchronization before presenting the data to the user, the major issue is to reduce the data transfer cost in accessing MDO's from different sites. This can be done by optimally allocating the MDO's to the sites of the distributed multimedia database system.

Optimal allocation of MDO's is a complex problem because of mutual interdependency between allocation scheme (which gives the location of each of the MDO's at various sites of a distributed multimedia database system) and query optimization strategy (which decides how a query can be

Manuscript received May 5, 1995; revised March 7, 1996.

The authors are with the Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (email: iahmad@cs.ust.hk).

Publisher Item Identifier S 0733-8716(96)06121-5.

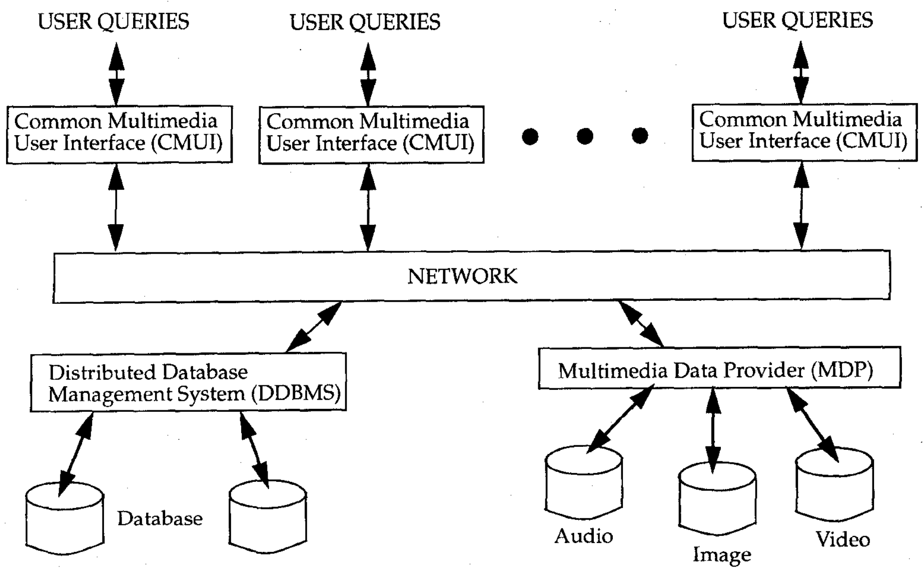

USER QUERIES
USER QUERIES
USER QUERIES
Common Multimedia User Interface (CMUI)
Common Multimedia User Interface (CMUI)
Common Multimedia User Interface (CMUI)
NETWORK
Distributed Database Management System (DDBMS)
Multimedia Data Provider (MDP)
Database
Audio
Image
Video

Fig. 1. Loosely coupled architecture of distributed multimedia database system.

optimally executed, given an allocation scheme). Given a query execution strategy, we model the dependencies between the MDO's and the amount of data transfer incurred to execute a query as site independent MDO dependency graph (MDG). We use the MDO dependency graph to solve for an optimal or near optimal allocation of MDO's. The processing strategy of distributed multimedia objects retrieval involves shipping of all the multimedia objects to the user's query site because this strategy supports efficient access for synchronization during the presentation of the result by the CMUI.

Fig. 2 shows the steps in data allocation strategy developed in this paper. Essentially, there are two aspects to this problem. The first aspect is to represent and evaluate the set of queries accessing the distributed multimedia database. The second aspect is to use this information to come up with a formulation and solution for the data allocation problem. The query processing [17] consists of decomposing the queries and data localization. Data localization involves identifying the MDO's accessed by the query and generating the MDO query operator trees. These MDO query operator trees are further processed by taking into consideration a given distributed query execution strategy and the information about the MDO sizes to generate MDO dependency graphs. The MDO dependency graph models the dependencies between the MDO's accessed by a query and the amount of data transfer incurred to execute a query. A data allocation algorithm takes as inputs: 1) the MDO dependency graphs, 2) unit data transfer costs between sites, 3) the allocation limit on the number of MDO's that can be allocated at a site, 4) the query execution frequencies from the sites, 5) the number of MDO's, and 6) the number of sites, and outputs an allocation scheme.

The objective of this work is to design efficient algorithms to generate minimum total data transfer cost allocation scheme. The rest of the paper is organized as follows: Section II further elaborates the data allocation problem. Section III describes a cost model used to calculate the total data transfer cost incurred to execute a set of queries. Section IV includes the algorithms proposed in this paper. The experimental results for these algorithms are provided in Section V. The effectiveness and suitability of the proposed algorithms to different scenarios is discussed in Section VI. The related work is presented in Section VII, and Section VIII concludes this paper.

II. The Data Allocation Problem

In this section, we describe in detail the inputs to the data allocation problem addressed in this paper. These inputs characterize the underlying distributed multimedia database system and help in formulating the problem. We also introduce a number of notations throughout the paper which are summarized in Table I.

Consider a distributed multimedia database system with m sites, with each site having its own processing power, memory and a database system. Let S_i be the name of site i where $0 \leq i \leq m-1$. The m sites of the distributed multimedia database system are connected by a communication network. A link between two sites S_i and $S_{i'}$ (if it exists) has a positive integer $c_{ii'}$ associated with it giving the cost for a unit data transferred from site S_i to site $S_{i'}$. If two sites are not directly connected by a communication link then the cost for unit data transferred is given by the sum of the cost of links of a chosen path from site S_i to site $S_{i'}$. Let $Q = \{q_0, q_1, \cdots, q_{n-1}\}$ be the most important queries accounting for say more than 80% of the processing in the distributed multimedia database system. Each query q_x can be executed from any site with a certain frequency. Let a_{ix} be the frequency with which query q_x is executed from site S_i. The executions frequencies of n queries at m sites can be represented by a $m \times n$ matrix, A. Let there be k MDO's (or database objects, or relations), named $\{O_0, O_1, \cdots, O_{k-1}\}$.

A. Query Representation

Any query accessing both the single-media database fragments and multimedia objects can be split into two queries, one which accesses only single-media fragments and one which

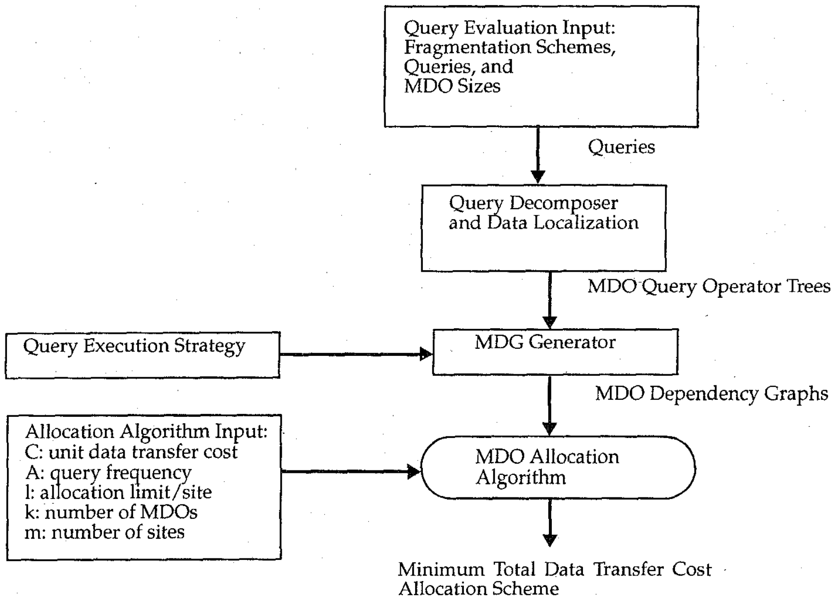

Query Evaluation Input: Fragmentation Schemes, Queries, and MDO Sizes
Queries
Query Decomposer and Data Localization
MDO Query Operator Trees
Query Execution Strategy
MDG Generator
MDO Dependency Graphs
Allocation Algorithm Input:
C: unit data transfer cost
A: query frequency
l: allocation limit/site
k: number of MDOs
m: number of sites
MDO Allocation Algorithm
Minimum Total Data Transfer Cost Allocation Scheme

Fig. 2. Steps in MDO allocation.

TABLE I
SYMBOLS AND THEIR MEANINGS

Symbol	Meaning
O_j	The jth MDO
S_i	The ith site
Q	The set of queries
q_x	The xth query
m	The number of sites in the network
k	The number of MDOs in the distributed database system
n	The number of queries
A	The access frequencies matrix
a_{ix}	The access frequency of the xth query at site i
C	The unit transportation cost matrix of the network
$c_{ii'}$	The unit transportation cost from site i to site i'
l	The allocation limit vector of the sites
l_i	The allocation limit of site i
R	The query data transfer size matrix
r_{xj}	The query data transfer size of the xth query of MDO j
U	The site data transfer size matrix
$u_{jj'}$	The site data transfer size of MDO j to site j'
D	The MDO dependency matrix
d_{ij}	The size of the data from MDO i to the site where MDO j is located
t	The total data transfer cost

accesses only multimedia data. The basis for this split is provided by the architecture of the loosely coupled distributed multimedia database system. The queries that access single media database fragments will be optimized by distributed database system, and the queries that access multimedia data will be processed by the multimedia data provider at the site.

As mentioned above, the data allocation problem is complex because of the mutual interdependency between the query execution strategy (decided by the query optimizer) and the allocation of the fragments. The optimal allocation of the fragments depends on a given query execution strategy while the optimal query execution strategy depends on a fixed materialization of the fragments (i.e., fixing the location of the fragments accessed by the query). The main problem in deciding on the optimal allocation is the lack of a representational model of the dependencies among the data fragments accessed by the query. These dependencies arise because of the partitioning of the relations into data fragments (by using methodologies proposed by [7]) and/or access to multiple relations by a query.

We use the distributed query decomposer and data localization algorithms [17] to decompose a distributed query into a set of queries. This decomposed query incorporates the dependencies between data fragments. These dependencies model binary operations (like join, union) between the data fragments that need to be processed in order to execute the distributed query. We estimate the sizes of the intermediate relations [17] generated after executing the unary and binary operations by using the database statistics (like, cardinality and lengths of tuples of the fragments/relations) available from the system catalog. In the distributed database query processing the optimal binary operation ordering is based on a fixed query execution strategy. By fixing the query execution strategy we eliminate the dependency between the distributed query execution strategy and data allocation. A query execution strategy can be the following:

1) *Move-Small:* If a binary operation involves two data fragments located at two different sites then ship the smaller data fragment to the site of the larger data fragment.
2) *Query-Site:* Ship all the data fragments to the site of query origin and execute the query.

Usually, the *move-small* query execution strategy is used by the distributed database system because it gives rise to lower data transfer costs for query execution than the *query-site* query execution strategy. As mentioned before, the MDP uses the *query-site* query execution strategy because of the synchronization requirements imposed on accessing the multimedia objects and the need to present each multimedia object as an individual data stream. Thus, the first aim of data allocation is to maximize the locality of the MDO's (consisting of single-media or multi media) for executing the queries. The second aim is to use the query execution strategy to reduce the total data transfer cost when a query needs to access MDO's from different sites.

B. Query Evaluation

Each of the n queries on the distributed multimedia database are restructured and decomposed to generate query operator trees.

Example 1: Consider the following query adapted from [17] accessing relations *E(Eno, Ename, Title)*, *G(Eno, Jno, Resp, Dur)*, and *J(Jno, Jname, Budget)* and multimedia objects representing the pictures *EPicture(Eno, Picture)* and the speech *ESpeech(Eno, Speech)* as shown in Fig. 3(a) and (b), respectively. For enhancing the readability, we use nested SQL statement to exemplify MDO retrieval specification. From the data allocation point of view, we need only consider the shipping of the retrieved Picture and Speech from EPicture and ESpeech, respectively, given the Eno values.

The relational algebra tree for the query in Fig. 3(a) after query decomposition and data localization is shown in Fig. 4(a). The intermediate relations generated after the query restructuring phase are J', G', E', G'', and J''. Assume that the size of G' is greater than the size of E', and the size of J' is greater than the size of G''. By using the move-small query execution strategy it is preferable to transfer E' to the

```
SELECT Ename
FROM J, G, E
WHERE G.Eno = E.Eno
AND G.Jno = J. Jno
AND Ename <> "J.Doe"
AND J.Name = "CAD / CAM "
AND (Dur = 12 OR Dur = 24);
```

(a)

```
SELECT Picture, Speech
FROM EPicture, ESpeech
WHERE Eno in (SELECT Eno
          FROM J, G, E
          WHERE G.Eno = E.Eno
          AND G.Jno = J. Jno
          AND Ename <> "J.Doe"
          AND J.Name = "CAD / CAM "
          AND (Dur = 12 OR Dur = 24));
```

(b)

Fig. 3. (a) SQL statement on distributed relational database system and (b) query statement to retrieve MDO's from EPicture and ESpeech.

site where G' is located and also to transfer G'' to the site where J' is located, in order to minimize the total size of data required to be transferred to execute the query. The move-small query execution strategy and the corresponding fragment dependency graph that is generated is illustrated in Fig. 4(b).

In a fragment dependency graph, the fragment-nodes (like *Site(J)*, *Site(G)*, etc.) represent the potential sites where the fragments are located. The query-node *Site(Q)* represents the site where the query is initiated (i.e., the query site, which is fixed for each execution of the query). There is a cost value attached to each edge of the graph corresponding to the amount of data that may be transferred if the fragments corresponding to the two nodes of the edge are located at different sites, or if the location of the fragment and the query site are different sites. The fragment dependency graph models the execution strategy used by the query optimizer without exactly fixing the locations of the fragments. This enables us to calculate the amount of data transfer incurred to process a query under different data allocation schemes. For example, referring to Fig. 4(b), if relations E and G are located at different sites then it will incur *Size(E')* data transfer cost to process the join between relations E' and G' when using *move-small* query execution strategy. Similarly, by applying the fragment dependency graph concept to multimedia data retrieval we generate the MDO data dependency graph accessing the speech and picture from ESpeech and EPicture, respectively, by using the *query-site* query execution strategy [see Fig. 4(c)].

III. The Data Transfer Cost Model

The cost model developed in this section is applicable for calculating the total data transfer cost incurred to process both single-media distributed database query and multimedia data retrieval query. We will not make any further distinction between single media database fragments and multimedia data for the rest of the paper, and will consider the problem in the context of MDO's—the retrieval of MDO's will be modeled by the MDO dependency graphs that can arise from either or both *move-small* or query site query execution strategy.

There are two aspects of the data transfer cost incurred to process a query that need to be modeled. The first aspect is the unit data transfer cost from one site to another. This is modeled as minimum cost path and the corresponding path from one site to another. We use an all-pairs shortest path algorithm to generate the cost matrix C, where c_{ij} is cost of transporting a single unit of data from site S_i to site S_j. Note that even if the network is fully connected the shortest path between

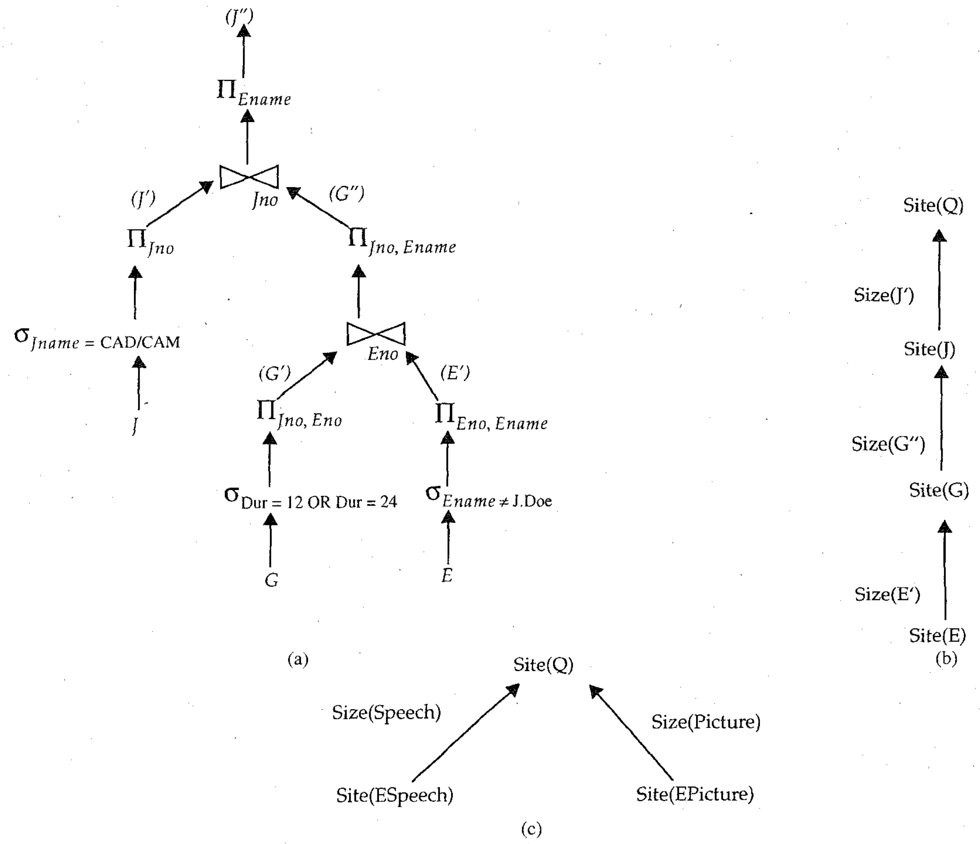


Fig. 4. (a) An example of fragment query operator tree, (b) the move-small fragment dependency graph, and (c) query-site MDO data dependency graph.

two sites may have more than one edge. In order to find the best allocation of a set of MDO's (i.e., the allocation which minimizes the total data transfer cost), it is enough to know the size of data from every MDO that is required from every site. This is the second aspect of data transfer cost that needs to be calculated. However, the problem is that, as represented in the MDO's dependency graphs, the size of data from a MDO that is required from a site varies with the locations of other MDO's. The MDO dependency graph of every query models two types of data transfer cost. The first type of cost is due to moving the data from the sites where the MDO's are located to the site where the query is initiated. The second type of cost is due to moving the data from the site where one MDO is located to the site where another MDO is located. For the first type of cost, the size of data of a MDO required by every site does not vary with the location of other MDO's as there is no dependency between the MDO's accessed by the query. This is true in the case of query-site query processing strategy, and the top level of the MDO dependency graphs. In the case of *move-small* query processing strategy [for example in Fig. 4(b)], the edge *Site(J)* $\rightarrow$ *Site(Q)* models this type of data transfer cost.

Let r_{xj} be defined as the size of data of MDO O_j needed to be transported to the site where q_x is initiated. The corresponding matrix is R of size $n \times k$. Note that this incorporates the top level of the MDO's dependency graph. Let there be a query q_x initiated from site S_i, a_{ix} times in an unit time interval. And let q_x request MDO O_j and each request require r_{xj} amount of data transfer from the site where O_j is located.

Let U be a $m \times k$ matrix where u_{ij} gives the amount of data needed to be transferred from the site where MDO O_j is allocated to the site S_i where the queries are initiated. That is

$$u_{ij} = \sum_{x=0}^{n-1} a_{ix} \cdot r_{xj}.$$

And in matrix representation

$$U = A \cdot R.$$

The second type of data transfer cost corresponds to the deeper levels of the MDO's dependency graph. The data is transported from the site where one MDO is located to the site where the other MDO is located in order to perform binary operation involving two (or more) different MDO's. In this case, the amount of data of a MDO required by a site varies with the allocation of other MDO's. Let $d_{jj'}$ define the size of data from MDO O_j that needs to be transported to the site where $O_{j'}$ is located so as to execute some binary operation. Let the corresponding matrix, $k \times k$, be D. But this is dependent on the query that is to be processed. Therefore, for each query we need to extract the information about how much data needs to be transferred from site where one MDO is located to the site where another MDO is located given that both the MDO's are accessed by the query. This information is extracted by the MDO's dependency graph generator which processes the query operator trees on MDO's by applying a query execution strategy and is represented in the MDO dependency graph.

Let $\delta^x_{jj'}$ be the data size of O_j needed to be transported to the site where $O_{j'}$ is located to process q_x. And let the corresponding matrix be ∇^x. Then the amount of data that needs to be transported from the site where O_j is located to the site where $O_{j'}$ is given by

$$d_{jj'} = \sum_{x=0}^{n-1}\left(\sum_{i=0}^{m-1} a_{ix}\right)\delta^x_{jj'}.$$

And in matrix representation

$$D = \sum_{x=0}^{n-1}\left(\sum_{i=0}^{m-1} a_{ix}\right)\nabla^x.$$

Let *site*(O_j) denote the site where MDO O_j is located. Then the total transportation cost, T, is given by

$$\sum_{j=0}^{k-1}\sum_{j'=0}^{k-1} c_{\text{site}(O_j),\text{site}(O_{j'})}\cdot d_{jj'} + \sum_{i=0}^{m-1}\sum_{j=0}^{k-1} u_{ij}\cdot c_{i,\text{site}(O_j)}.$$

where the first term gives the data transfer cost incurred to process the binary operations between the MDO's located at different sites, and the second term gives the data transfer cost incurred to transfer the results of the binary operations of MDO's to the site where the query is initiated. The objective in data allocation problem is to minimize T by altering the function *site*(O_j) (which maps a MDO to a site).

IV. Proposed Data Allocation Algorithms

Developing an efficient solution to the data allocation problem highly depends on the query execution strategy employed by the distributed database system. This is because different query execution strategies have different MDO migration patterns. The data allocation algorithm is NP-complete in general [11]. In this section, we develop solutions for the allocation problem when *query-site* and *move-small* query execution strategies are respectively used by the distributed database management system and the multimedia data provider. As will be described below, optimal solution can be obtained in polynomial time if the query execution strategy is constrained to be *query-site* only.

In this section, we present three approaches to tackle the data allocation problem using four different algorithms. In the first approach, we employ a hill-climbing algorithm to generate a near-optimal solution. In the second approach, we formulate the problem as a state-space search problem and solve it by using a best-first search algorithm. The third approach is a graph partitioning approach and is based on two clustering techniques.These algorithms are explained below.

A. Optimal Data Allocation: Max-Flow Min-Cut Approach

We first describe an algorithm that generates optimal solutions for query-site query execution strategy. We use Floyd-Warshall's algorithm [26] to find the minimal cost communication path between any two sites. This gives us the $c_{ij'}$ values in the cost matrix C. The running time of this algorithm is $O(m^3)$ where m is the number of sites in the distributed database system.

In this case, all the MDO's are transferred to the query site, and the query is executed at the site of its origin. This is the same as neglecting the dependency among MDO's and allocating the MDO's when *move-small* query execution strategy is used. Let MDO O_j be allocated at S_i then let W be a $m\times k$ matrix where w_{ij} represents the cost of data transferred from O_j to site S_i (i.e., the second term in expression for the total transportation cost). That is

$$w_{ij} = \sum_{i'=0}^{m-1} c_{ii'}\cdot u_{i'j}.$$

And in matrix representation,

$$W = C\cdot U.$$

Let X_{ij} be 1 if O_j is allocated at site S_i, and zero otherwise. The allocation problem is formulated as assigning zero-one values to X_{ij} under the constraints,

1) $\sum_{i=0}^{m-1} X_{ij} = 1, \quad \forall 0 \le j \le k-1.$
2) $\sum_{j=0}^{k-1} X_{ij} \le l_i, \quad \forall 0 \le i \le m-1.$

so as to minimize

$$\sum_{i=0}^{m-1}\sum_{j=0}^{k-1}(1 - X_{ij})\cdot w_{ij}.$$

The first constraint ensures that each MDO is allocated to at least one site, and the second constraint ensures that no site is allocated more MDO's than the maximum number that can be allocated at that site.

Based on the above formalism, the data allocation problem in this case can be viewed as a mapping problem. Given the costs of data transfer incurred (i.e., w_{ij}) when a MDO O_j is allocated at site S_i, the problem is to map the MDO's to sites so as to minimize the total data transfer cost. Since a single site will not have enough storage space for all the MDO's there is a limit on number of MDO's that can be allocated at a site. This problem is equivalent to a maximum-flow minimum-cost problem, and an optimal mapping can be achieved.

The outline of the solution is formulated as follows. Calculate W as shown previously. Then, transform the problem into a maximum-flow minimum-cost problem. After the costs of allocating MDO's at each site have been calculated (W is found), the problem of allocating MDO's to sites in order to get minimum data transfer cost can be viewed as a maximum-flow minimum-cost problem. The first step in translating allocation problem to a maximum-flow minimum-cost problem is to calculate the cost of allocating a MDO to a particular site. This is done by calculating the matrix W.

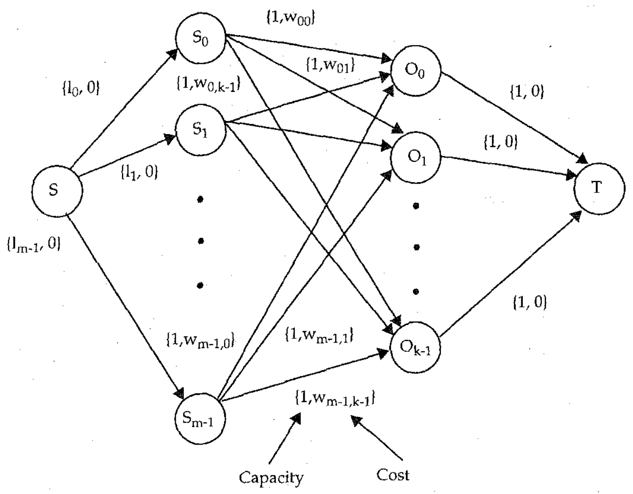


Fig. 5. Formulating the data allocation problem to max-flow min-cut problem.

In order to formulate the data allocation problem as a maximum-flow minimum-cut problem (see Fig. 5) we need to perform following steps.

1) Two nodes, a source and a sink, are created and named, S and T, respectively.
2) m nodes are created, one each corresponding to a site named, $\{S_0, S_1, \cdots, S_{m-1}\}$.
3) k nodes are created, one each corresponding to a MDO named, $\{O_0, O_1, \cdots, O_{k-1}\}$.
4) For each node S_i, where $0 \leq i \leq m-1$, an edge from S to it is created and the capacity and cost of the link are assigned as l_i and 0, respectively.
5) For each node O_j, where $0 \leq j \leq k-1$, an edge from it to T is created and the capacity and cost of the link are assigned to 1 and 0, respectively.
6) For every pair of nodes S_i and O_j, an edge from S_i to O_j is created and the capacity and cost of the link are assigned as 1 and w_{ij}, respectively.

The problem is to find the maximum flow with minimum cost from S to T. It can be observed that the maximum flow in this case is k (i.e., the number of MDO's in the distributed database system), since it is bounded by the sum of capacity of incoming edges to T. The sum of capacity of outgoing edges from S must be bigger than k (the number of MDO's) for all MDO's to be allocated. This will be true if $\sum_{i=0}^{m-1} l_i \geq k$. At the point of maximum flow, there must be exactly one incoming edge (say from node S_i) to O_j with flow equal to 1 and all other incoming edges from nodes S_p have flow equal to zero, for $0 \leq p \leq m-1$ and $p \neq i$. This is equivalent to allocating MDO O_j to site S_i. Since the capacity of flow from S to site S_i is assigned $l_i, 0 \leq i \leq k-1$, no site will be allocated more than the maximum limit on the number of MDO's allowed. The problem then is to achieve maximum flow from node S to node T with the minimum cost. This is equivalent to assigning each MDO to a site while minimizing the total data transfer cost.

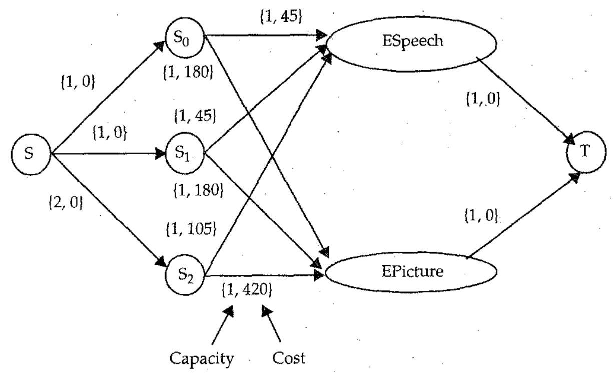


Fig. 6. Max-flow min-cut formulation for the example allocation problem.

Lemma: The maximum-flow and minimum-cost formulation of data allocation problem with *query-site* query execution strategy generates the optimal solution.

Proof: The data allocation problem can be restated as a mapping problem (i.e., mapping a MDO to a site). When the *query-site* query execution strategy is used, the MDO's dependency graph of queries has only one level. This implies that the data transfer cost incurred to execute all the queries when a MDO is located at a site is fixed (i.e., independent of the allocation of other MDO's). That is, all the values for $d_{jj'}$ are zero, for $0 \leq j, j' \leq k-1$. The data transfer cost for the MDO O_j to be allocated on site S_i can be found by summing up $\sum_{i'=0}^{m-1} \sum_{j'=0}^{n-1} (c_{ii'} \cdot a_{i'j'} \cdot r_{j'j})$, which is exactly the values of the elements of matrix W. These are the cost values on the edge from site S_i to MDO O_j in Fig. 5. Therefore, by finding the maximum flows of this graph, a feasible allocation is found. And by finding the minimum-cost maximum-flow a minimum total data transfer cost allocation is found, which implies that the optimal solution is achieved. The value of the minimum total data transfer cost is the sum of the costs w_{ij} on the edges S_i from to O_j which form the maximum-flow and minimum-cost solution. □

Example 2: Consider a distributed multimedia database system with 3 fully connected sites S_0, S_1, and S_2, and two MDO's *ESpeech*, and *EPicture*. Let there be only one query (as shown in Fig. 4) accessing the multimedia service provider. The MDO dependency graph is illustrated in Fig. 4(c). Let this query be initiated from site S_0 with frequency 3, from site S_1 with frequency 2, and from site S_2 with frequency 1. Let the sizes of intermediate MDO's be: *size(ESpeech)* $= 5$, and *size(EPicture)* $= 20$. Since there is only one query, the matrix $R = [5 \;\; 20]$ corresponding to MDO's *ESpeech*, and *EPicture*, respectively, and the matrix $A = [3 \;\; 2 \;\; 1]^t$ (i.e., a column vector) corresponding to sites S_0, S_1, and S_2 respectively. Let the limit vector l, constraining the number of MDO's that can be allocated to a site be, $l = [1 \;\; 1 \;\; 2]$. Then the matrix $U = A \cdot R$ is as follows (the rows correspond to sites S_0, S_1, and S_2, respectively, and the columns correspond to MDO's *ESpeech*, and *EPicture* respectively)

$$U = \begin{bmatrix} 15 & 60 \\ 10 & 40 \\ 5 & 20 \end{bmatrix}.$$

Let the cost matrix for unit data transfer cost from one site to another be (the rows and columns correspond to sites S_0, S_1, and S_2, respectively)

$$C = \begin{bmatrix} 0 & 2 & 5 \\ 2 & 0 & 3 \\ 5 & 3 & 0 \end{bmatrix}.$$

Then the total data transfer from a MDO to a site is given by the matrix $W = C \cdot U$ as follows (the rows correspond to sites S_0, S_1, and S_2 respectively, and the columns correspond to MDO's *ESpeech*, and *EPicture*, respectively)

$$W = \begin{bmatrix} 0 & 2 & 5 \\ 2 & 0 & 3 \\ 5 & 3 & 0 \end{bmatrix} \cdot \begin{bmatrix} 15 & 60 \\ 10 & 40 \\ 5 & 20 \end{bmatrix} = \begin{bmatrix} 45 & 180 \\ 45 & 180 \\ 105 & 420 \end{bmatrix}.$$

Fig. 6 illustrates the max-flow min-cut formulation for this example. After applying the Ford and Fulkerson method [26] to solve for maximum-flow with minimum cost, we get the solution as: allocate E to site S_0, and J to site S_1. The total data transfer cost is: $45 + 180 = 225$.

B. The Hill-Climbing Approach

The general data allocation problem has been proved to be NP-complete [11]. Finding the optimal solution by exhaustive search would require $O(k^m)$ in the worst case where k is the number of MDO's and m is the number of sites. Therefore, a heuristic algorithm based on hill-climbing technique is developed in order to find a near optimal solution. The general data allocation problem solution consists of following two steps:

1) As a first step, neglect the dependencies among the MDO's, and come up with a initial data allocation, by using the max-flow min-cut formulation of the problem (as described in Section IV-A).
2) Iteratively improve the initial data allocation by using the hill climbing heuristic until no further reduction in total data transfer cost can be achieved. This is done by applying some operations on the initial allocation scheme. Since there are finite number of feasible allocations, the heuristic algorithm will complete its execution.

The initial solution generated by the max-flow min-cut formulation is refined by applying some operations on it. The objective for applying these operations is to reduce the total data transfer cost. There are two types of operations that are defined, namely, *migrate* (migrate MDO's from its currently allocated site to the newly allocated site), and *swap* (swap the locations of one set of MDO's with the locations of new set of MDO's). Note that a migrate operation cannot be applied if the limit on the number of MDO's that can be located at a site is exceeded. Whereas, the swap operation does not change the number of MDO's allocated at each of the sites. These operations take into consideration the dependency between the MDO's as modeled by MDO dependency graphs. In order to come up with a general solution, these operations are independent of exact structure of a MDO dependency graph but are based on the type of processing involved in executing a query in a distributed fashion. These operations are iteratively applied until no more reduction is observed in the total data transfer cost. Fig. 7 shows the major steps in the hill-climbing heuristic algorithm that is developed in this section.

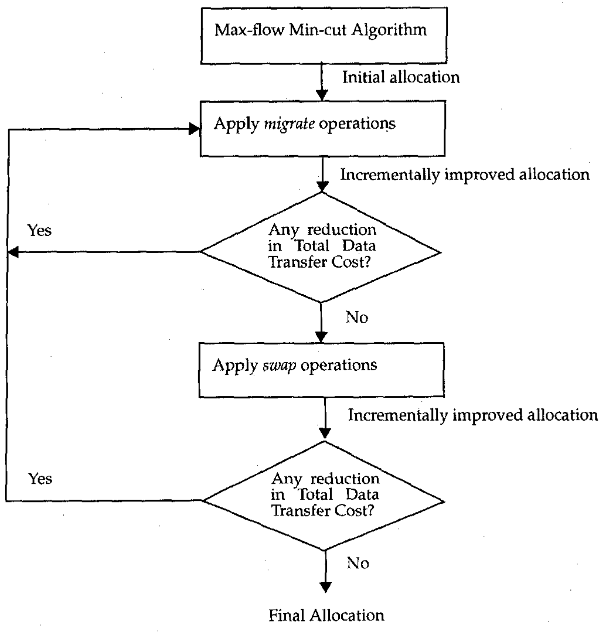

Max-flow Min-cut Algorithm
Initial allocation
Apply *migrate* operations
Incrementally improved allocation
Any reduction in Total Data Transfer Cost?
Yes
No
Apply *swap* operations
Incrementally improved allocation
Any reduction in Total Data Transfer Cost?
Yes
No
Final Allocation

Fig. 7. Steps in the hill-climbing algorithm for data allocation.

The set of *migrate* and *swap* operations are as follows.

1) *Migrate* (O_j, S_i): move MDO O_j to S_i. This operation can be applied to each MDO, and the MDO can be moved to any one of $m-1$ sites (it is not located at). Therefore, there can be maximum of $k(m-1)$ *migrate* operations that can be applied during each iteration. The *migrate* operation can potentially reduce the data transfer cost involved in processing the binary operation over two MDO's. If one MDO is located at one site and another MDO is located at a different site, then migrating one MDO to the site of another can potentially reduce data transfer cost.
2) *Migrate2* $(O_j, S_i, O_{j'}, S_{i'})$: move MDO O_j to site S_i and MDO O'_j to site S'_i. This operation can be applied to each pair of MDO's, and they can each be moved to any one of $m-1$ sites where they are not located. Therefore, there can be maximum of $(k(k-1)/2)(m-1)^2$ *migrate2* operations that can be applied during each iteration. As in the case of migrate operation, the *migrate2* operation can potentially reduce data transfer costs in processing binary operations involving three MDO's.
3) *Migrate3* $(O_j, S_i, O_{j'}, S_{i'}, O_{j''}, S_{i''})$: move MDO O_j to site S_i, MDO $O_{j'}$ to site $S_{i'}$ and MDO $O_{j''}$ to site $S_{i''}$. This operation can be applied to each triplet of MDO's, and they can each be moved to any one of $m-1$ sites where they are not located. Therefore, there can be a maximum of $(k(k-1)(k-2)/6)(m-1)^3$ *migrate3* operations that can be applied during each iteration. The *migrate3* operations can potentially reduce data transfer costs in processing binary operations over four MDO's.
4) *Swap* $(O_x, O_{x'})$: swap the location of MDO O_x with

the location of MDO $O_{x'}$. This operation can be applied to each distinct pair of MDO's. Therefore, there can be a maximum of $k(k-1)/2$ *swap* operations that can be applied during each iteration. The *swap* operation is useful in minimizing the total data transfer cost by monitoring the data transfer between two sites. Since there is a limit on number of MDO's that can be allocated at a site, by swapping MDO's we maintain this constraint, and simultaneously explore if we can improve the allocation.

5) *Swap2* $(O_x, O_{x'}, O_y, O_{y'})$: swap the location of MDO O_x with the location of MDO $O_{x'}$, and location of O_y with location of $O_{y'}$. This operation can be applied to each of the distinct pairs of MDO's O_x, $O_{x'}$ and O_y, $O_{y'}$. The *swap2* operation is also useful in reducing the total data transfer cost while maintaining the constraint on number of MDO's that can be allocated to a site.

6) *Swap3* $(O_x, O_{x'}, O_y, O_{y'}, O_z, O_{z'})$: swap the location of MDO O_x with the location of MDO $O_{x'}$, location of O_y with location of $O_{y'}$, and location of O_z with the location of $O_{z'}$. This operation can be applied to each of the distinct pairs of MDO's O_x, $x_{x'}$, O_y, $O_{y'}$ and O_z, $O_{z'}$. The *swap3* operation is also useful in reducing the total data transfer cost while maintaining the constraint on number of MDO's that can be allocated to a site.

In the experiments that we have conducted, we noticed that *migrate* and *swap* operations are applied most often, followed by *migrate2* and *swap2* operations. The least frequent occurrences were that of *migrate3* and *swap3* operations. In our extensive experimental studies we found that many times just applying *migrate* and *swap* operations would not give the global optimal solution, but would give a global solution when *migrate2* and/or *swap2* operations, or *migrate3* and/or *swap3* operations were applied. This was the reason for us to define three different types of migrate and swap operations. The run time complexity was higher for the hill-climbing algorithm to use *migrate4* or *swap4* operations.

Example 3: We use the example presented in Section II-B with the distributed database system using the *move-small* query execution strategy. The MDO dependency graph is as illustrated in Fig. 4(b). Let the MDO sizes are: *size*$(E') = 5$, *size*$(J') = 30$, and *size*$(G') = 25$. First, we use the top-level of the MDO dependency graph; i.e., the edge from *Site*$(J) \rightarrow$ *Site*(Q), to solve for initial allocation scheme by using maximum-flow minimum cut formulation. Let this query be initiated from site S_0 with frequency 3, from site S_1 with frequency 2, and from site S_2 with frequency 1. As there is only one query, we have the matrix $R = [0 \ \ 0 \ \ 30]$ with each element corresponding to relations E', G', and J', respectively, and the matrix $A = [3 \ \ 2 \ \ 1]^t$ (i.e., a column vector) with each element corresponding to sites S_0, S_1, and S_2, respectively. Let the limit vector l, constraining the number of MDO's that can be allocated to a site be, $l = [2 \ \ 2 \ \ 2]$. Let the cost matrix for unit data transfer cost from one site to another be (the rows and columns correspond to sites S_0, S_1, and S_2, respectively)

$$C = \begin{bmatrix} 0 & 2 & 5 \\ 2 & 0 & 3 \\ 5 & 3 & 0 \end{bmatrix}.$$

After formulating the problem as maximum-flow and minimum cut problem and solving it (as illustrated in Section IV-A) we get the initial solution as: allocate MDO E at site S_0, allocate MDO G at site S_1, and allocate MDO J at site S_0. This allocation schema is represented as $\{S_0, S_1, S_0\}$ corresponding to the sites where the relations E, G, and J are to be located, respectively.

The matrix ∇^1 giving the amount of data needed from a MDO to be transported to site of another MDO derived from the MDO dependency graph [shown in Fig. 4(c)] is

$$\nabla^1 = \begin{bmatrix} 0 & 5 & 0 \\ 0 & 0 & 25 \\ 0 & 0 & 0 \end{bmatrix}.$$

The row and columns of matrix ∇^1 correspond to MDO's E', G', and J', respectively. We get the total amount of data that must be transported from the site where one MDO is located to the site where another MDO is located as (the rows and columns correspond to MDO's E', G', and J', respectively)

$$D = (3+2+1) \cdot \begin{bmatrix} 0 & 5 & 0 \\ 0 & 0 & 25 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 0 \\ 0 & 0 & 150 \\ 0 & 0 & 0 \end{bmatrix}.$$

The total data transfer cost given for the initial allocation $\{S_0, S_1, S_0\}$ is $(2 \times 30 + 2 \times 150) + (2+1) \times 30 = 450$. The cost value in first parenthesis corresponds to the data transfer cost incurred in transferring E' to site where G is located, and transferring G'' to the site where J is located. The second cost value corresponds to the data transfer incurred in transferring the result J' to the query site.

We now apply the hill-climbing heuristic algorithm to improve upon the initial solution so as to further reduce the total data transfer cost if possible. Table II illustrates the working of hill-climbing algorithm.

Table II shows the migrate and swap operations applied to improve the initial solution provided by the max-flow min-cut algorithm. Two operations were applied, 1) *migrate*(J, S_1) (which reduced the total data transfer cost from 450 to 180), and 2) *migrate*(E, S_1, G, S_0, J, S_0) (which reduced the data transfer cost from 180 to 150). The solution to data allocation problem generated by the hill-climbing algorithm is $\{S_1, S_0, S_0\}$ (i.e., allocate relation E at site S_1, relation G at site S_0, and relation J at site S_0) with the total data transfer cost incurred to execute the query as 150. Table III shows all feasible allocation schemes and the total data transfer cost incurred for each of them. Table III also shows the itemized costs for the data transfer. The cost involved in transferring E' to site where relation G is located is given by the column *Size*$(E') \rightarrow$ *Site*(G), the column *Size*$(G'') \rightarrow$ *Site*(J) gives the cost of transferring G'' to the site where relation J is located, and the column *Size*$(J') \rightarrow$ *Site*(Q) gives the cost of

TABLE II
OUTPUT OF THE HILL-CLIMBING ALGORITHM FOR DATA ALLOCATION

Current Allocation	Operation Type	Operation Applied	New Allocation	Total Data Transfer Cost
$\{S_0, S_1, S_0\}$	*migrate*	*migrate* (J, S_1)	$\{S_0, S_1, S_1\}$	180
$\{S_0, S_1, S_1\}$	*migrate*	None applied	—	—
$\{S_0, S_1, S_1\}$	*migrate2*	None applied	—	—
$\{S_0, S_1, S_1\}$	*migrate3*	*migrate* (E, S_1, G, S_0, J, S_0)	$\{S_1, S_0, S_0\}$	150
$\{S_1, S_0, S_0\}$	*migrate*	None applied	—	—
$\{S_1, S_0, S_0\}$	*migrate2*	None applied	—	—
$\{S_1, S_0, S_0\}$	*migrate3*	None applied	—	—
$\{S_1, S_0, S_0\}$	*swap*	None applied	—	—
$\{S_1, S_0, S_0\}$	*swap2*	None applied	—	—
$\{S_1, S_0, S_0\}$	*swap3*	None applied	—	—

transferring the J' to the query site. Following points can be noted after comparing hill-climbing algorithm with the results of exhaustive solution.

1) The hill-climbing algorithm generates the optimal solution (which is allocation number 9) in Table III.
2) The hill-climbing algorithm would not have find the optimal solution if only *migrate* operation had been used. Note that only after applying *migrate3* operation was optimal solution achieved.
3) The hill-climbing algorithm may not always generate an optimal solution

C. BFS/ESP Algorithm

In this section, the data allocation problem is formulated as a state-space search problem. A best-first search algorithm, which is based on similar principles as the A* algorithm, is used for solving the problem [25]. In a state-space search problem, each state description is denoted by a node. Operators applicable to nodes are defined for generating successors of nodes, called node expansion. A solution path of a search problem is a path in the state-space defined by a sequence of operators which leads a start node to one of the goal nodes. The A* algorithm has been successfully applied to the tasks to processors mapping problem in parallel processing. In the data allocation problem, a solution path defines an allocation which has the minimum cost. The state-space search problem is formalized as follows.

1) *State Description:* Let a set of ordered pairs of MDO and site denote the partially developed allocation corresponding to a node in the search tree.
2) *Initial State:* The initial state is the empty set.
3) *Operators:* An operator adds a new pair of MDO and site to the current set under node expansion. Every unallocated MDO-site pair is a candidate.
4) *Goal State:* The goal state is reached if every MDO is allocated to a site.

The above formulation just offers a search scheme for finding a solution. We use an algorithm called best-first search with efficient search-space pruning (BFS/ESP) to find the optimal solution. The BFS/ESP algorithm is based on similar principles as the A* algorithm in that it uses an evaluation function to judge which search node to expand next. In an A* algorithm, an evaluation function is used to order nodes for expansion, and is guaranteed to find a optimal solution path in that the path cost is minimized if the evaluation function is admissible. More specifically, we define an evaluation function as

$$f(N) = g(N) + h(N)$$

where $g(N)$ is the minimum path cost from the start node to node N in the state-space, and $h(N)$ is a lower-bound estimate, using any heuristic information available, of the minimum path cost $h^*(N)$ from node N to a goal node. As long as $h(N) \leq h^*(N)$ for all N (i.e., $h(N)$ is consistent), evaluation function is called admissible and the A* algorithm using such an evaluation function will be guaranteed to find an optimal solution path in the state space after expanding fewer nodes during the search than any uninformed algorithms. That is, such a heuristic search algorithm can speed up the search of an optimal data allocation, which is usually time consuming for graphs with large numbers of MDO's and sites. In order to speed up the search process by properly pruning the search space, the heuristic function has to be tight. If its value is too small (e.g., zero), then the search degenerates to an exhaustive search. If its value is too large, then it may not be admissible and cannot guarantee optimal solution. Thus, such heuristic function is problem-dependent and is usually difficult to define.

For the data allocation algorithm, $g(N)$ of a search node N is simply the cost of the partial allocation. This partial cost can be easily computed by using the total transfer cost

TABLE III
ENUMERATION OF FEASIBLE ALLOCATION SCHEMES AND THE RESPECTIVE DATA TRANSFER COSTS

Allocation Number	Allocation Scheme	Data Transfer Cost (Itemized)			Total Data Transfer Cost
		$Size(E') \to Site(G)$	$Size(G'') \to Site(G)$	$Size(J') \to Site(Q)$	
1	$\{S_0, S_0, S_1\}$	0	2×150	$(3+1)\times30$	420
2	$\{S_0, S_0, S_2\}$	0	5×150	$(3+2)\times30$	900
3	$\{S_0, S_1, S_0\}$	2×30	2×150	$(2+1)\times30$	450
4	$\{S_0, S_1, S_1\}$	2×30	0	$(3+1)\times30$	180
5	$\{S_0, S_1, S_2\}$	2×30	3×150	$(3+2)\times30$	660
6	$\{S_0, S_2, S_0\}$	5×30	5×150	$(2+1)\times30$	990
7	$\{S_0, S_2, S_1\}$	5×30	3×150	$(3+1)\times30$	720
8	$\{S_0, S_2, S_2\}$	2×30	0	$(3+2)\times30$	210
9	$\{S_1, S_0, S_0\}$	2×30	0	$(2+1)\times30$	150
10	$\{S_1, S_0, S_1\}$	2×30	2×150	$(3+1)\times30$	480
11	$\{S_1, S_0, S_2\}$	2×30	5×150	$(3+2)\times30$	960
12	$\{S_1, S_1, S_0\}$	0	2×150	$(2+1)\times30$	390
13	$\{S_1, S_1, S_2\}$	0	3×150	$(3+2)\times30$	600
14	$\{S_1, S_2, S_0\}$	3×30	5×150	$(2+1)\times30$	930
15	$\{S_1, S_2, S_1\}$	3×30	3×150	$(3+1)\times30$	620
16	$\{S_1, S_2, S_2\}$	3×30	0	$(3+2)\times30$	240
17	$\{S_2, S_0, S_0\}$	5×30	0	$(2+1)\times30$	240
18	$\{S_2, S_0, S_1\}$	5×30	2×150	$(3+1)\times30$	570
19	$\{S_2, S_0, S_2\}$	5×30	5×150	$(3+2)\times30$	1050
20	$\{S_2, S_1, S_0\}$	3×30	2×150	$(2+1)\times30$	480
21	$\{S_2, S_1, S_1\}$	3×30	0	$(3+1)\times30$	210
22	$\{S_2, S_1, S_2\}$	3×30	3×150	$(3+2)\times30$	750
23	$\{S_2, S_2, S_0\}$	0	5×150	$(2+1)\times30$	840
24	$\{S_2, S_2, S_1\}$	0	3×150	$(3+1)\times30$	570

expression with the unallocated MDO's being excluded from the expression. As to the heuristic function $h(N)$, it can be defined by several different approaches. The simplest way is to set $h(N) = 0$ for all N, and the resulting search is a uniform-cost search. To be more efficient in the search, nonzero $h(N)$ should be used. In our approach, we define $h(N)$ as follows.

```
1)  Let $O$ be the set of unallocated MDO's at the current
    search node $N$.
2)  Set $h(N) = 0$.
3)  FOR each MDO $o \in O$ DO
4)    Set $MinCost=MAXINT$.
5)    FOR each site $s$ DO
6)      Determine $ThisCost$ to be the additional allocation
        cost if $o$ is allocated to $s$.
7)      IF $ThisCost \le MinCost$ THEN Set $MinCost = ThisCost$.
8)    ENDFOR
9)    $h(N) = h(N) + MinCost$.
10) ENDFOR
```

Using the above procedure, however, it is not difficult to see that $h(N)$ is not always a consistent lower bound of $h^*(N)$ for all N. In other words, the search algorithm cannot guarantee optimal solutions in all cases. Given the definitions of $g(N)$ and $h(N)$, the BFS/ESP algorithm for solving the data allocation problem is briefly described below.

BFS/ESP Data Allocation Algorithm:

1) Put the initial search node $I = \phi$ on a list called OPEN, and set $f(I) = 0$.
2) Remove from OPEN the search node n with the smallest f, and put it on a list called CLOSED.
3) If N is a goal node, then determine the final allocation and stop; otherwise, go to the next step.
4) Expand search node n by considering the allocation of the next MDO to all the sites. Compute $f(N') =$

$g(N') + h(N')$ for each successor node N' of N. Put all the successors on OPEN. Go to step 2).

In the worst case, the BFS/ESP algorithm may explore the entire search tree before the best solution is located. Thus, the worst case time complexity of the BFS/ESP algorithm is $O(m^k)$. However, it should be noted that with the above-mentioned heuristic node evaluation function, the BFS/ESP algorithm can take much less time than the worst case time complexity implies [10].

D. Agglomerative Clustering

In this section, we describe two clustering approaches to tackle the data allocation problem. In general, search-based methods can produce better solutions at the expense of much longer execution time, while clustering approaches are much faster but may generate less optimal solutions [3], [16], [23], [24]. Clustering algorithms can be broadly divided into two classes [4].

1) *Agglomerative Algorithms:* The graph to be clustered is initially considered to have N unit clusters. A number of clustering operations are then performed to produce a certain number of larger clusters.
2) *Divisive Algorithms:* The graph is initially considered to be a single cluster. The large cluster is then incrementally cracked to form a number of smaller clusters.

In this section, we present an agglomerative algorithm that performs clustering of the MDO's followed by a mapping of the clusters to the sites. In this algorithm, the MDO's are considered to be connected as a clique. Each edge in the clique is associated with a weight, denoted by $e_{ii'}$, representing the aggregate amount of data dependence between the two MDO's i and i' across all the queries. Each node in the clique represents the maximum allocation cost of the MDO's, denoted by w_i (determined by the *query-site* strategy), across all sites. Given the MDO dependency graphs and the queries set, this MDO clique can be computed before the clustering algorithm starts. The clustering algorithm works by first locating a *pivot* MDO, defined as the MDO that is connected by the heaviest edge, and then incrementally includes neighbor MDO's until the cumulative maximum MDO allocation costs exceeds a threshold T. This threshold is simply the sum of all the maximum MDO allocation costs divided by the number of sites in the system. Following the clustering of the pivot MDO, the same procedure is applied to a new pivot MDO selected from the remaining unclustered MDO's.

After a set of MDO clusters are formed, each cluster is mapped to a site. The criterion of choosing a site for a MDO cluster is simple: the site that gives the least increase in the partial allocation cost is the target site. The algorithm is briefly presented below.

Agglomerative Clustering:

```
1) Construct the MDO clique by:
   For each MDO $i$, compute the maximum allocation cost
   $i$ across all sites. This value, denoted by $w_i$, is taken as
   the node weight of $i$ in the clique.
   For each pair of MDO $i$ and $i'$, determine the aggregate
   MDO dependency costs between $i$ and $i'$ across all
   queries. This cost is denoted by $e_{ii'}$ and is taken as
   the edge weight in the clique.
2) Determine a threshold $T$, defined as the sum of all
   weights divided by the number of sites.
3) WHILE not all MDO's are clustered DO
4)    Determine the pivot MDO, defined as the MDO with
      the heaviest edge.
5)    Incrementally include neighbor MDO's in order of
      decreasing edge weight until the total node weights
      exceeds $T$.
6) ENDWHILE
7) Set TotalCost $=$ 0.
8) FOR each MDO cluster $C$, in decreasing order of total
   weights DO
9)    Map $C$ to a site $s$ such that the increase in the partial
      allocation cost, $\Delta$Cost, is the minimum.
10)   TotalCost $=$ TotalCost $+ \Delta$Cost.
11) ENDFOR
```

Compared with the previous approaches, the AC clustering algorithm incurs much less time complexity. The time complexity of the AC algorithm is only $O(k^2m(k+m))$.

E. Divisive Clustering

The agglomerative clustering (AC) algorithm, described above, is designed to be a MDO *dependency-oriented* algorithm. That is, we put the main emphasis on the dependency among MDO's in determining the target data allocation. In this section, we present a divisive clustering (DC) algorithm, which is designed to be a *site-oriented* algorithm. That is, we consider the network communication costs among the sites as more important factor in determining a better allocation. More specifically, the DC algorithm works by first allocating all the MDO's to a *pivot* site. The pivot site is the one that has the lowest aggregate communication costs among all the sites. The sum of all the communication overhead of the outgoing links of the pivot site to other sites is the minimum among all the sites. Each MDO is then considered to be migrated to another site which can lower the cost of the allocation. All sites are considered for the migration of each MDO. The process terminates when all the MDO's are examined. Similar to the AC algorithm, the design of this DC algorithm is again simple so that it admits very efficient implementation. The algorithm is briefly described below.

Divisive Clustering:

```
1) Find out the site that has the lowest aggregate communi-
   cation overhead across all channels. Let it be the pivot
   site.
2) Allocate all the MDO's to the pivot site. Determine the
   cost of the allocation.
3) FOR each MDO $o$ in the pivot site DO
4)    Set BestSite $=$ pivot, $\Delta$Cost $=$ 0.
5)    FOR each other site $s$ DO
6)       Compute $\Delta$Cost$'$ if $o$ is migrated to $s$.
7)    IF $\Delta$Cost$' \leq \Delta$Cost THEN set BestSite $= s$,
      $\Delta$Cost $= \Delta$Cost$'$.
```

TABLE IV
EXPERIMENTAL RESULTS OF THE HILL-CLIMBING ALGORITHM

No. of Sites	No. of MDOs	No. of Problems	No. of Opt. Sol.	Aver. % Deviation	Number of Sol. with Deviation. in range (0, 5%)	[5,10%)	[10,20%)	[20%, -)
5	5	100	95	1.5832	5	0	0	0
5	6	100	82	1.1149	18	0	0	0
5	7	100	87	1.0243	13	0	0	0
5	8	100	89	0.7539	11	0	0	0
6	5	100	93	0.6412	7	0	0	0
6	6	100	95	0.8160	5	0	0	0
6	7	100	84	0.9123	16	0	0	0
6	8	100	81	0.9228	19	0	0	0
7	5	100	88	0.6100	12	0	0	0
7	6	100	86	0.8796	14	0	0	0
7	7	100	88	0.8161	12	0	0	0
7	8	100	76	0.8264	24	0	0	0
8	5	100	87	0.9841	13	0	0	0
8	6	100	88	0.4505	12	0	0	0
8	7	100	83	0.7225	17	0	0	0
8	8	100	83	0.9034	16	1	0	0

```
8)  ENDFOR
9)  Migrate $o$ to BestSite.
10) ENDFOR
```

Similar to the AC algorithm, the time complexity of the DC algorithm is also much less than the search based methods. The time complexity of the DC algorithm is $O(k^2m(k+m))$.

V. RESULTS

In this section, we present the experimental results for the data allocation algorithms described in the previous sections. Comparisons among these algorithms will be made by considering the quality of solutions and the algorithm running times.

A. Workload

The example considered in the previous section was used for illustrating how the hill-climbing algorithm works. But it had only one query, three MDO's and three sites. The number of feasible allocations were only 24, with data transfer cost ranging from 150 to 1050. Since the solutions were to be compared with the optimal solutions generated by exhaustive search and the exhaustive search method takes large amount of time to experiment for a distributed multimedia database system with even moderate number of sites and MDO's (for k MDO's and m sites there are k^m allocation schemes, and for each allocation scheme the data transfer cost needs to be calculated), we conducted sixteen experiments with number of MDO's ranging from five to eight, and number of sites ranging from five to eight. Each experiment consisted of 100 allocation problems with number of sites and number of MDO's fixed. Each allocation problem had between 10 and 20 queries, and each query had a MDO dependency graph. The communication network, the relation sizes, the link costs, and the structure of MDO dependency graph were randomly generated from a uniform distribution. Each data allocation algorithm described above was tested for every case and statistics were collected.

B. Comparison of Allocation Costs

In Tables IV to VII, we list, for each of the experiments conducted in a column-wise fashion, the following: 1) the sites, 2) number of MDO's, 3) number of problems, 4) number of problems for which the hill-climbing algorithm generated the optimal solution, 5) the average deviation in percentage of near optimal solutions from optimal solution for those allocation problems for which the hill-climbing algorithm did not generate optimal solution, 6) number of near optimal solutions with deviation less than 5%, 7) number of near optimal solutions with deviation of 5% or more but less than 10%, 8) number of near optimal solutions with deviation of 10% or more but less than 20%, and 9) number of near optimal solutions with deviation 20% or more. The number of optimal solutions can reflect how good the algorithm is; whereas the average deviation shows how bad the algorithm performs when it cannot generate optimal solutions.

From Table IV, we note that the hill-climbing algorithm generated optimal solution for a large number of problems—1385 cases out of a total of 1600 cases, corresponding to about 86% of the test cases. In addition, the overall average deviation (i.e., the mean of all average deviations) of a near optimal solution from the optimal solution was only about 0.87% in total data transfer cost. Furthermore, for the tests that when the hill-climbing algorithm generates suboptimal solutions, almost all of them are within 5% from the optimal. Thus, the hill-climbing approach is a very effective algorithm in that it can generate optimal solutions for most cases and very near to optimal solutions when it cannot generate optimal solution.

Table V includes the results of the BFS/ESP algorithm. Although not as good as the hill-climbing approach, the BFS/ESP algorithm performs well. The number of optimal solutions is 926 out of 1600—more than 57%. Moreover,

TABLE V
EXPERIMENTAL RESULTS OF THE BFS/ESP ALGORITHM

No. of Sites	No. of MDOs	No. of Problems	No. of Opt. Sol.	Aver. % Deviation	Number of Sol. with Deviation. in range (0, 5%)	[5,10%)	[10,20%)	[20%, -)
5	5	100	69	5.6043	24	6	0	1
5	6	100	62	3.8830	25	7	6	0
5	7	100	57	5.1204	31	6	4	2
5	8	100	47	3.7522	40	8	4	1
6	5	100	71	3.5797	23	3	3	0
6	6	100	63	2.3129	35	1	1	0
6	7	100	61	3.2448	31	6	1	1
6	8	100	50	2.8268	38	10	2	0
7	5	100	68	1.8634	30	2	0	0
7	6	100	59	2.0448	36	5	0	0
7	7	100	55	2.1835	41	2	2	0
7	8	100	58	1.9695	37	5	0	0
8	5	100	69	1.4174	29	2	0	0
8	6	100	52	2.3270	41	6	1	0
8	7	100	45	1.5727	53	1	1	0
8	8	100	40	1.6548	56	4	0	0

TABLE VI
EXPERIMENTAL RESULTS OF THE AGGLOMERATIVE CLUSTERING ALGORITHM

No. of Sites	No. of MDOs	No. of Problems	No. of Opt. Sol.	Aver. % Deviation	Number of Sol. with Deviation. in range (0, 5%)	[5,10%)	[10,20%)	[20%, -)
5	5	100	66	5.6070	25	8	0	1
5	6	100	43	3.4601	43	7	7	0
5	7	100	49	4.4600	38	8	3	2
5	8	100	47	3.9008	38	10	4	1
6	5	100	67	3.3800	26	4	3	0
6	6	100	48	2.2310	48	3	1	0
6	7	100	49	3.5626	43	4	3	1
6	8	100	37	2.9107	50	10	3	0
7	5	100	63	1.9947	35	2	0	0
7	6	100	50	2.0075	46	4	0	0
7	7	100	44	1.9855	52	4	0	0
7	8	100	40	2.1333	53	6	1	0
8	5	100	65	1.8618	30	5	0	0
8	6	100	44	2.3550	49	6	1	0
8	7	100	36	1.7192	60	3	1	0
8	8	100	38	1.7162	58	4	0	0

more than 80% of the remaining cases are only 5% from the optimal. Only five out of 1600 cases are more than 20% from the optimal. The overall average deviation from optimal is small—about 2.83%. The BFS/ESP algorithm is more efficient than the hill-climbing approach because it takes 2 order of magnitude less time to generate a solution, as will be seen in the following subsection.

Table VI contains the performance data of the agglomerative clustering (AC) algorithm. The performance of the AC algorithm is comparable to that of the BFS/ESP algorithm. Despite that the number of optimal solutions generated is fewer (786 out of 1600) the number of solutions with costs within 20% from the optimal is the same as that of the BFS/ESP algorithm. This can be noted from the table that both algorithms have only five cases out of 1600 with costs more than 20% from the optimal. The overall average deviation is also similar—about 2.83%. Based on these results, we can notice that the agglomerative clustering algorithm is indeed very cost-effective. This is because all the search based algorithms take exponential time in the worst case.

Table VII contains the performance results of the divisive clustering (DC) algorithm. The performance of the DC algorithm is again comparable to both the AC and BFS/ESP algorithms. The number of optimal solutions generated is 802 out of 1600. And almost all of the suboptimal solutions are also within 20% from the optimal. The overall average deviation is also similar—about 2.82%. Thus, we can observe that for the data allocation problem, there is no particular preference to either the agglomerative approach or the divisive approach.

C. *Comparison of Running Times*

Table VIII contains the average running times of all algorithms for each experiment. For comparison, the time taken to generate the optimal solutions by using exhaustive search

TABLE VII
EXPERIMENTAL RESULTS OF THE DIVISIVE CLUSTERING ALGORITHM

No. of Sites	No. of MDOs	No. of Problems	No. of Opt. Sol.	Aver. % Deviation	Number of Sol. with Deviation. in range (0, 5%)	[5,10%)	[10,20%)	[20%, -)
5	5	100	62	4.9888	31	6	0	1
5	6	100	61	3.3866	29	5	5	0
5	7	100	42	4.2882	43	10	3	2
5	8	100	40	3.7411	42	14	3	1
6	5	100	67	3.4530	27	4	2	0
6	6	100	55	2.5675	40	3	2	0
6	7	100	48	3.2486	42	8	1	1
6	8	100	40	2.6330	50	8	2	0
7	5	100	63	1.9495	35	2	0	0
7	6	100	49	1.8516	47	4	0	0
7	7	100	41	2.5595	53	4	1	1
7	8	100	48	2.2369	46	5	1	0
8	5	100	64	1.4758	34	2	0	0
8	6	100	44	2.9364	42	12	2	0
8	7	100	39	1.9406	56	3	2	0
8	8	100	39	1.8989	57	4	0	0

TABLE VIII
AVERAGE RUNNING TIMES (MSECS) OF ALL THE ALGORITHMS

No. of Sites	No. of MDOs	Exhaustive Search	Hill-Climbing	BFS/ESP	AC	DC
5	5	110.55	120.92	4.46	2.68	2.55
5	6	679.53	259.29	9.86	4.80	4.81
5	7	4111.79	487.48	15.46	7.90	8.01
5	8	24044.91	961.96	34.88	12.65	13.12
6	5	325.98	206.27	6.06	3.27	3.57
6	6	2344.89	421.09	13.35	5.66	6.26
6	7	25599.84	1339.67	34.90	15.12	16.94
6	8	169081.30	2018.21	56.27	20.80	24.70
7	5	648.65	273.16	6.40	3.45	3.72
7	6	5667.73	597.79	14.47	5.95	6.54
7	7	45137.12	1152.67	34.36	9.67	10.32
7	8	440446.44	2272.42	62.37	15.84	18.40
8	5	1447.89	458.87	9.67	4.16	4.84
8	6	14915.47	898.44	22.97	7.54	8.05
8	7	134759.01	1940.40	36.40	11.83	13.81
8	8	1312713.03	3377.72	49.46	19.25	22.44

are also listed. All the algorithms were implemented on a SPARC IPX workstation and the time data were measured in milliseconds. As can be seen from the table, although the hill-climbing approach took much short time compared with exhaustive search, it took two order of magnitude more time than the clustering algorithms, and the BFS/ESP algorithm. The two clustering algorithms were found to be the fastest algorithms. Indeed, they are the more cost-effective algorithms when compared against the other algorithms.

VI. Discussion

From the experiment results presented in the previous section, we observe that the clustering algorithms are very cost-effective if fast execution is desired. Therefore, these algorithms can be applied to generate allocation schemes for limited redesign of distributed multimedia databases [27]. Limited redesign implies change in allocation scheme but not in fragmentation scheme for a distributed database. The algorithms developed in this paper can be integrated with algorithms for affecting the limited redesign of distributed databases [22] and thus facilitating on-line tuning of a distributed multimedia database system. If solution quality is the more prominent factor, the hill-climbing approach is a viable choice for an off-line allocation. The BFS/ESP algorithm, whose solution quality is next to that of the hill-climbing approach, is also a practical approach, though it can take much longer time than the clustering algorithms. The clustering algorithms can be useful if the quality of the solution is not the main objective but the algorithm running times is the major consideration.

VII. Related Work

The data allocation problem has been first studied in terms of file allocation problem in a multi computer system, and later on as a data allocation problem in distributed database system. The file allocation problem does not take into consideration the semantics of the processing being done on files, whereas data allocation problem must take into consideration the interdependencies among accesses to multiple fragments by a query. Chu [8] studied the problem of file allocation with respect to multiple files on a multiprocessor system. He presented a global optimization model to minimize overall processing costs under the constraints of response time and storage capacity with a fixed number of copies of each file. Casey [5] distinguished between updates and queries on files. Eswaran [11] proved that Casey's formulation is NP complete. He suggested that a heuristic rather than exhaustive search approach is more suitable.

Ramamoorthy and Wah [21] analyzed a file allocation problem in the environment of a distributed database and developed a heuristic approximation algorithm for a simple file allocation problem and the generalized file allocation problem. They also proposed a model for file migration or reallocation that is identical in formulation to the file allocation problem. Ceri *et al.* [6] considered the problem of file allocation for typical distributed database applications with a simple model for transaction execution taking into account the dependencies between accesses to multiple fragments.

Apers [1] considered the allocation of the distributed database to the sites so as to minimize total data transfer cost. The author came up with a complicated approach to allocate relations by first partitioning them into innumerable number of fragments, and then allocating them. The author did integrate the system dependent query processing strategy with the logical model for allocating the fragments. But the author solves the complete fragmentation and allocation problem. That is, the fragmentation schema is one of the outputs of the allocation algorithm. This curtails the applicability of this methodology when fragmentation schema is already defined and allocation scheme must be generated.

Cornell and Yu [9] proposed a strategy to integrate the treatment of relation assignment and query strategy to optimize performance of a distributed database system. Though they took into consideration the query execution strategy, the solution they came up with is a complicated linear programming solution. The main problem in their approach is the lack of simplicity in both incorporation of the query execution strategy and the solution procedure.

There have been many linear programming formulations proposed for data allocation problem [12], [20]. The main problem with these approaches is the lack of modeling of the query execution strategy. Lin *et al.* [14] also developed a heuristic algorithm for minimum overall data transfer cost, by considering replicated allocation of fragments and both read and update transactions.

VIII. CONCLUSION

The problem of nonredundant data allocation of MDO's in distributed database system is addressed in this paper. A query driven data allocation approach is developed by integrating the query execution strategy with the formulation of the data allocation problem. We studied the data allocation problem for two popularly used query execution strategies, namely, *query site* and *move-small.* We formulated the data allocation problem with query-site execution strategy as a max-flow min-cut problem and showed that it can be solved to generate an optimal solution. A hill-climbing heuristic algorithm was developed for the data allocation problem with move-small query execution strategy. We studied the effectiveness of this algorithm by conducting a set of experiments, which show that the hill-climbing heuristic algorithm generates the optimal solution for a large percentage of allocation problems. Moreover, the near optimal solution generated by the algorithm has on an average small percentage of deviation from the optimal solution in terms of the total data transfer costs incurred to process all the queries. The problem was also formulated as a state-space searched problem and was solved with a efficient search algorithm called BFS/ESP. Two clustering algorithms were also proposed which have much shorter execution times although the quality of solutions are slightly worse than the hill- climbing approach. The proposed data allocation algorithms can be used in practice for allocating MDO's in distributed multimedia database systems.

A major contribution of this work is the development of a site-independent MDO dependency graph representation of data transfer costs incurred in executing a query by using a particular query execution strategy. This enables the application of query driven allocation approach to allocate database objects under different database systems (like, relational, object-oriented, network, etc.) and facilitates the development of an uniform framework within which to define and solve data allocation problems. A complicated query execution strategy employed by a distributed database system will generate complex MDO dependency graphs which will involve development of novel techniques to solve for optimal allocation.

REFERENCES

[1] P. M. G. Apers, "Data allocation in distributed database systems," *ACM Trans. Database Syst.*, vol. 13, no. 3, pp. 263–304, Sept. 1988.

[2] P. B. Berra, C. Y. R. Chen, A. Ghafoor, C. C. Lin, T. D. C. Little, and D. Shin, "Architecture for distributed multimedia systems," *Comput. Commun.*, vol. 13, no. 4, pp. 217–31, May 1990.

[3] M. J. Berger and S. H. Bokhari, "A partitioning strategy for nonuniform problems on multiprocessors," *IEEE Trans. Comput.*, vol. COMP-36, no. 5, pp. 570–580, May 1987.

[4] N. S. Bowen, C. N. Nikolaou, and A. Ghafoor, "On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems," *IEEE Trans. Comput.*, vol. 41, no. 3, pp. 257–273, Mar. 1992.

[5] R. G. Casey, "Allocation of copies of a file in an information network," in *Proc. Spring Joint Comput. Conf.*, IFIPS, 1972, pp. 617–625.

[6] S. Ceri, G. Martella, and G. Pelagatti, "Optimal file allocation for a distributed on a network of minicomputers," in *Proc. Int. Conf. Database*, Aberdeen, July 1980, pp. 345–357.

[7] S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*. New York: McGraw Hill, 1984.

[8] W. W. Chu, "Optimal file allocation in a multiple computer system," *IEEE Trans. Comput.*, vol. COMP-18, no. 10, 1969.

[9] D. W. Cornell and P. S. Yu, "Site assignment for relations and join operations in the distributed transaction processing environment," in *Proc. Int. Conf. Data Engineering*, IEEE, Feb. 1988.

[10] S. Dutt and N. R. Mahapatra, "Parallel A* algorithms and their performance on hypercube multiprocessors," in *Proc. Int. Parallel Processing Symp.*, Apr. 1993, pp. 797–803.

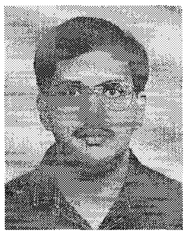
[11] K. P. Eswaran, "Placement of records in a file and file allocation in a computer network," *Informat. Processing*, pp. 304–307, 1974.

[12] B. Gavish and H. Pirkul, "Computer and database location in distributed computer systems," *IEEE Trans. Comput.*, vol. COMP-35, no. 7, pp. 583–590, 1986.
[13] A. Ghafoor, "Multimedia database management systems," *ACM Comput. Surveys*, vol. 27, no. 4, pp. 593–598, Dec. 1995.
[14] X. M. Lin, M. E. Orlowska, and Y. C. Zhang, "Database placement in communication networks for minimizing the overall transmission cost," *Mathematical Comput. Modeling*, vol. 19, no. 1, pp. 7–19, Jan. 1994.
[15] T. D. C. Little and A. Ghafoor, "Synchronization and storage models for multimedia objects," *IEEE J. Select. Areas Commun.*, vol. 8, no. 3, pp. 413–427, Apr. 1990.
[16] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Trans. Comput.*, vol. 37, no. 11, pp. 1384–1397, Nov. 1988.
[17] T. Ozsu and P. Valduriez, *Principles of Database Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1991.
[18] M. T. Özsu, D. Szafron, G. El-Medani, and C. Vittal, "An object-oriented multimedia database system for a news-on-demand application," *Multimedia Syst.*, vol. 3, nos. 5/6, 1995.
[19] T. C. Rakow, E. J. Neuhold, and M. Lohr, "Multimedia database systems: The notions and the issues," to be published in *Tagungsband GI-Fachtagung Datenbanksystems* in Buro, Technik and Wissenschaft (BTW), Dresden Marz 1995, Springer Informatik Aktuell, Berlin, 1995.
[20] S. Ram and R. E. Marsten, "A model for database allocation incorporating a concurrency control mechanism," *IEEE Trans. Knowledge Data Eng.*, vol. 3, no. 3, pp. 389–395, Sept. 1991.
[21] C. V. Ramamoorthy and B. Wah, "The placement of relations on a distributed relational database," in *Proc. 1st Int. Conf. Distributed Comp. Syst.*, Huntsville, AL, pp. 642–649, Oct. 1979.
[22] P. I. Rivera-Vega, R. Varadarjan, and S. B. Navathe, "Scheduling data redistribution in distributed databases," in *Proc. Int. Conf. Data Eng.*, Feb. 1990.
[23] P. Sadayappan, F. Ercal, and J. Ramanujam, "Cluster partitioning approaches to mapping parallel programs onto a hypercube," *Parallel Comput.*, vol. 13, pp. 1–16, 1990.
[24] P. Sadayappan and F. Ercal, "Nearest-neighbor mapping of finite element graphs onto processor meshes," *IEEE Trans. Comput.*, vol. COMP-36, no. 12, pp. 1408–1424, Dec. 1987.
[25] C. C. Shen and W. H. Tsai, "A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion," *IEEE Trans. Comput.*, vol. COMP-34, no. 3, pp. 197–203, Mar. 1985.
[26] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, vol. CBMS-NSF44, 1983.
[27] B. Wilson and S. B. Navathe, "An analytical framework for the redesign of distributed databases," in *Proc. 6th Adv. Database Symp.*, Tokyo, Japan, 1986.

Yu-Kwong Kwok (S'94) received the B.Sc. degree in computer engineering from the University of Hong Kong, in 1991, and the M.Phil. degree in computer science from the Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, in 1994, where he is now a Ph.D. candidate.

His research interests include algorithms, parallel and distributed computing, programming languages, and compilers.

Mr. Kwok is a member of the ACM and the IEEE Computer Society.



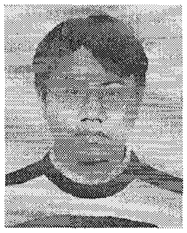
Kamalakar Karlapalem received the Ph.D. degree in computer science from the Georgia Institute of Technology, Atlanta, in 1992, the M.Tech. degree in computer science from the Indian Institute of Technology, Kharagpur, in 1986, and the MStat from the Indian Statistical Institute, in 1985.

He joined the Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, as an Assistant Professor, in 1993. Prior to his doctoral study, he worked for two years as a Software Engineer at Tata Consultancy Services and the National Informatics Centre, Government of India. He also worked as a Student Intern at Hewlett-Packard, Cupertino, CA, on conceptualization, design, and implementation of a distributed relational database system. His research interests are distributed database systems, database design, database system performance evaluation, distributed object management, workflow systems, cooperative problem solving, and data mining.

Ishfaq Ahmad (S'88–M'91) received the B.Sc. degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1985, the M.S. degree in computer engineering and the Ph.D. degree in computer science, both from Syracuse University, NY, in 1986 and 1992, respectively.

Currently, he is a Faculty Member in the Department of Computer Science at the Hong Kong University of Science and Technology, Clear Water Bay, Kowloon. His research interests include various aspects of parallel and distributed computing, high performance computer architectures and their assessment, performance evaluation, and distributed multimedia systems. He combines his hobbies with research by applying high-performance computers to hi-fi audio/video processing in his Video Technology Lab.

Dr. Ahmad received the Best Student Paper Awards at Supercomputing '90 and Supercomputing '91.



Ng Moon Pun received the M.S. degree in computer science from the Hong Kong University of Science and Technology, Clear Water Bay, Kowloon.

His research interests are computational geometry, data structures, and distributed database systems.