Query Plans - Their Cost and Generation in a Distributed Heterogeneous Environment

George Semeczko
Centre for Cooperative Information Systems
Queensland University of Technology
Brisbane, QLD 4001
Australia
georges@fit.qut.edu.au

Stanley Y.W. Su
Database Systems Research & Development Center
University of Florida
Gainesville, FL 32611
su@cis.ufl.edu

September, 1997

Abstract

As with other distributed database systems, it is important in an Active Object Oriented Knowledge Base Management System (AOOKBMS) to be able to execute distributed queries efficiently and cost effectively. In this paper, we show how this task is being tackled in an ongoing project at the University of Florida dealing with Virtual Enterprises (VE). This project uses an AOOKBMS to not only describe a common view of the VE's data and processing capabilities, but to also support distributed query optimization and processing. Through the use of wrappers this system allows the distributed execution of a query defined in an Object Query Language. It can be executed on heterogeneous systems that may include Object Oriented and Relational databases as well as non-database oriented systems. The results of this distributed processing is then combined and presented in a single uniform result, thus supporting the concept of a truly Virtual Enterprise. In this paper we cover how query optimization can be achieved using the AOOKBMS to store and manipulate all aspects of the query optimization and distributed processing tasks. In this paper we show that the AOOKBMS is a suitable tool for this task. To do this we present a cost function for any query tree defined in the AOOKBMS as well as an algorithm for turning a centralized query tree into a distributed query plan. Both this algorithm as well as the execution control mechanism are then presented using the implementation programming language developed for the AOOKBMS..

1. Introduction

In large organizations it is often necessary to perform complex computing tasks across many distributed and heterogeneous systems. Often these tasks are being carried out by many organizations which form a Virtual Enterprise (VE) in order to complete some mutually desirable goal. Not only are the computer systems to be connected often heterogeneous, but they are usually legacy systems. This restricts the degree to which they may be directly altered to aid in the goals of the VE. Coordination of the services and data available in such a system is often achieved by the use of traders or binders, Coulouris [9]. This allows for the various legacy systems' services to be advertised with appropriate protocols. Other systems can then locate the desired services and utilize the advertised interface specifications to access them. Unfortunately, the complex interactions between the systems in the VE often make programming of VE-wide operations extremely difficult using such mechanisms.

This problem has been approached in the National Industrial Information Infrastructure Protocols (NIIIP) project by using an Active Object-oriented Knowledge Base Management System (AOOKBMS). The AOOKBMS not only acts as a trader through an easily queried specification of the data and methods available in the VE, but also provides a very powerful and simple-to-use query language and an active rule processor.

The AOOKBMS being used at the University of Florida for this project is the OSAM*.KBMS. It is based on a semantic association model described by Su in [19][20][24] and as stated is active, Su [21]. An Object Query Language (OQL) is provided and described in detail in Alashqur [1] and Potharaju [5]. This system is highly extensible and has been modified to execute VE-wide queries in a distributed manner. This involves not only extensions to the meta-schema which describes the VE to handle this distributed processing as described in Semeczko [16] but also mechanisms to aid the distributed processing. One such mechanism is the migration facilities that are described in Semeczko [18]. Such mechanisms for handling distributed processing will not be useful unless they can be used effectively.

The objective of this paper is to describe the query optimization requirements for this system as well as presenting an initial algorithm for converting a centralized query plan in the form of a query tree into a distributed query execution plan. How this is implemented along with the query execution control mechanism is also presented using the programming language developed for the AOOKBMS in use.

This paper is structured as follows. In Section 2 several background areas are covered for completeness. It includes background on the AOOKBMS structure and query language facilities it provides as well as background on query optimization. Section 3 continues this work by discussing a cost function for a distributed query plan. This is followed by the presentation of a simple algorithm for turning a centralized query plan in the form of a query tree into a distributed query execution plan. In Section 4 the implementation of this simple algorithm and its execution control mechanism are presented. These are implemented using the programming language developed for the AOOKBMS in use at the University of Florida. This is then followed by Section 5 that contains concluding remarks, which include an update on the current state of the extensions to this system at the University of Florida.

2. Background

In this section, several background areas are discussed. The first section describes the current AOOKBMS model and system used for the VE. Following from this will be a discussion on the current OQL language, its capabilities and how it is to be processed in a distributed environment. This is then followed by a description of some of the existing work in the area of query optimization with the objective of describing some of the aspects that any query optimizer may need to consider.

2.1 Current AOOKBMS Architecture

One of the objectives of the NIIIP project is to facilitate the coordinated operations being performed within a VE that consists of many heterogeneous legacy systems. The objective is not to restrict interaction between these legacy systems, but to provide assistance when requested. Thus the approach taken in the NIIIP project was to use an Active Object-oriented Knowledge Base Management System (AOOKBMS) to assist the VE operations.

Each of these legacy systems interacts with this AOOKBMS via a wrapper. The wrappers perform any conversions necessary from and into the common global object-oriented model used in the AOOKBMS. In this way, any legacy system could communicate with another via this common object representation. These legacy systems may communicate directly with each either through these wrappers or via some other agreed protocols. Each of these possibilities is represented in Figure 1.

The OOKBMS used in this system is called the OSAM*.KBMS as described by Su in [21]. It is highly extensible and based on a semantic association model called OSAM* as described in Su [20][24]. Its active capabilities are described in Su[21]. Details of the architecture of the OSAM*.KBMS can be found in Su [21].

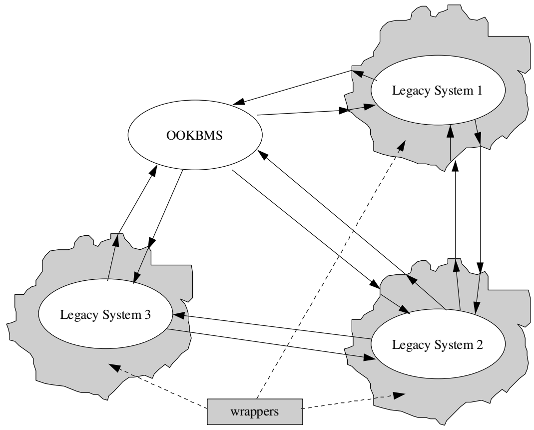

Figure 1:Use of common model by wrappers

Being an active object oriented system, it is used to describe the data and services offered to the VE by each legacy system in an object oriented manner. Not only does it act as a trader for the VE, through the provision of its query language, it provides a mechanism to define complex VE-wide operations using a single easy to use language. Details on the OQL can be found in Alashqur [1] and Potharaju [15] with some explanations provided in Section 2.2 of this paper.

The ability to perform these VE-wide operations in a distributed manner has been made possible through the extensions described in Semeczko [16] [18]. In Semeczko [18] the protocols that allow object migration to be performed were described. They were defined using the Knowledge Query and Manipulation Language (KQML) described in Finin [10]. These protocols were necessary for the work in Semeczko [16] to allow the distributed processing of queries. In this later paper extensions to the meta-model and query processing were described.

The meta-model used is shown in Figure 2. As described in Semeczko [16] the primary components of the model are classes and the associations between them. Classes are defined by attributes (associations with domain classes), methods and rules. As with most object-oriented models the concepts of generalization (labelled by a G) and aggregation (labelled by an A) are catered for. Here though they are included with the other possible associations between classes. They provide a much more rich foundation to describe relationships between objects. These include Generalisation (inheritance), Aggregation and Interaction. Other work at the University of Florida have added to these three using a model extensibility technique, but for the purposes of this paper, this basic set is sufficient. Examples of Aggregation and Generalization can be seen in the meta-model below. For example, the class *Class* is a generalization of the classes *Entity* and *Domain. Class* has two aggregation associations with domain classes (i.e. two attributes) and four aggregation associations with four entity classes. The two string attributes are *schema* and *name*. The other four aggregation associations are with the classes *Site, Method*, *Rule* and *Assoc*. The last three being of the set type. This allows for each class definition to contain a name, the schema it belongs to, a set of methods, a set of rules[1], a set of associations with other classes, and where a class is defined. Some of these aspects are now discussed a little further.

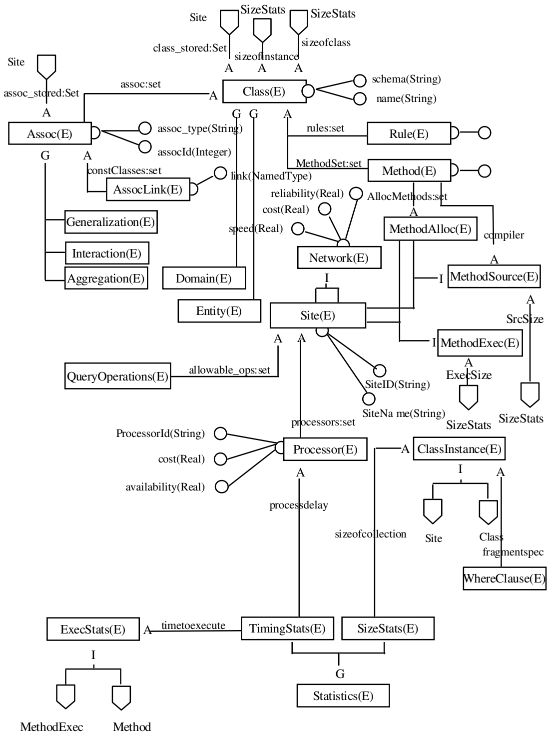

Figure 2-The New Meta-Model

In the meta-model above, the class *Assoc* also has a set association with the class *AssocLink*. This shows that an association can be defined by many links. *Assoc* is the generalization of the association classes *Generalization, Aggregation* and *Interaction*. For a detailed explanation of the association types see Su [20].

The migration protocols mentioned earlier are supported in this meta-model through the classes *MethodAlloc*, *MethodSource* and *MethodExec*. These three classes allow for the definition of executable methods and their components and sources. This can then be used to migrate the method components throughout the VE. See Semeczko [18] for further details.

Not shown in this meta-model is the fact that an Instance Identifier (IID) identifies every object instance stored in the AOOKBMS. This is made up of two components: Object Identifier (OID) and a class identifier. In this way, given an object instance, the KBMS not only can uniquely identify the object, but also the class to which the object belongs.

Statistical data used for query optimization purposes are also included. This covers not only size information but also processing performances and network capabilities. It is general enough to not only describe a distributed system but also a parallel processing system.

Information regarding the objects and their components includes: the location of objects and their components and the location of association data[2]. For some query optimization methods, information regarding the size of object components may also be needed. This data may be generated using statistical data.

Another aspect that is very important in regard to the complexities of the OQL language and its ability to be used in a VE, is operation processing ability. That is, for each site, which OQL operations can be performed at that site should be identified. It need not be specified how these operations are to be performed, but only that they can be performed. This caters for situations where the wrappers are able to transform OQL query operations into a locally equivalent operation.

To allow for fragmentation of classes and their distribution, a *ClassInstance* class has been included along with a horizontal fragmentation descriptor in the *WhereClause* class.

While not complete for all possible forms of query optimization, this model is sufficient to show the capabilities of the OSAM* model to describe the meta-data needed for query optimization and distributed processing. As the meta-model is easily extended, future requirements can be easily accommodated.

2.2 OQL and Query Processing

In this section, the capabilities of the current OQL language are presented. As well as this a description of the current design for a distributed query processing system is presented. It was originally shown in Semeczko [16] and has been implemented at the University of Florida [26]. An OQL query, as described in Alashqur [1] and Potharaju [15] takes the following form:

CONTEXT *association pattern expression*
[WHERE *conditions*]
[SELECT *object classes*]
DO *object method(s)*

Space limitations restrict the discussion on OQL queries, so only a brief description is possible here of the features available. In the CONTEXT clause, an association pattern is expressed by detailing classes and the association operators between them. Such operations include: association (*), non-association (!), compliment (|) as well as branching operations of "and" and "or".

1 Rules in this model are Event-Condition-Action-AlternativeAction (ECAA) rules.

2 - It is assumed that under normal circumstances that associations between classes stored at the same site will be stored locally. For associations that span legacy systems and hence sites, will be stored in the OSAM*.KBMS. This is consistent with it being used to store common and shared data.

The instances that match this pattern and form the result subdatabase can then be restricted further by the optional WHERE clause. This works in a similar manner to the WHERE clause in an SQL query. It is also possible to have WHERE restrictions based on the results of method calls.

Another optional component of an OQL query is the SELECT clause. This clause is also similar to that in an SQL statement except that instead of projecting on attributes, the projection occurs on classes. That is, this clause eliminates classes from the result subdatabase.

Finally, the DO components of an OQL query specify what operations are to be performed on the result subdatabase. These operations maybe either system-defined maintenance operations (e.g. InsertObject) or they may be user-defined operations defined as methods belonging to the objects.

An example of an OQL query from Alashqur [1] is presented:

CONTEXT Faculty * Section *
  and (Course * Department,RA)
  WHERE Department.name = ‘EE’
  SELECT Faculty [name], c#
DO display

This query can be used to answer the following request for a university database:

“Display the name of any faculty member who is teaching any section of a course that is offered by the ‘EE’ department, provided that the section is taken by at least one graduate student who is an RA. Also show the course number.”

To process the OQL queries in a distributed manner a new Distributed Query Processor (DQP) was described in Semeczko [16]. Its architecture is shown in Figure 3.

The Query Parser verifies the OQL query and creates an internal structure for use in the Query Tree (QT) construction by the QT Transformer.

The QT had two major subtrees: Context and Operation. This QT is different to earlier centralized query processor. In the centralized version, the context component was subdivided into three subtrees: association pattern expressions, WHERE clauses, and the SELECT clauses. In the distributed version, it is essential to construct subqueries that contain elements of each of these components and then assign their evaluation to remote sites. For this reason, this second subdivision of the QT has been dropped in the new QT structure. The schema shown in Figure 4 defines the design for this new QT.

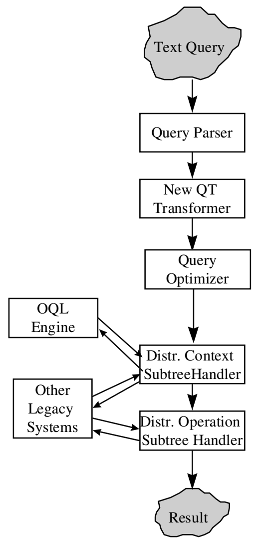

Figure 3- Distributed Query Processing Architecture

In this new QT, four types of nodes in the tree can be identified. Each node represents one of four types of operations to be performed: instance access, unary operations, binary operations, and do operations. First, the generalized node definition is discussed and then its specializations covered.

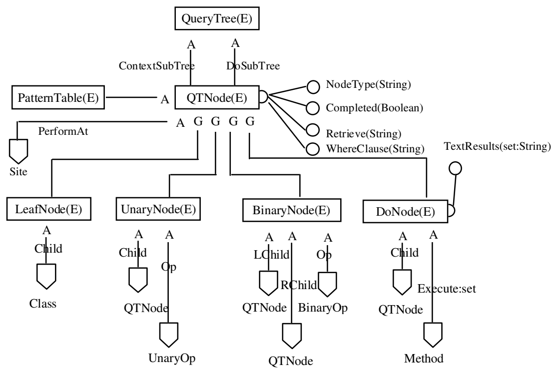

Figure 4- New Query Tree Definition

For each node a descriptor, *NodeType*, is used to specify the type of node and whether the operation that it pertains to has been *Completed* or not. If it has been completed then the node is then associated with a *PatternTable*. This represents the subdatabase that is the result of all operations in the subtree that has this node as its root. Its association to a Site instance specifies the location where the operation represented by this node is to be performed. Also, as nearly all operations can be coupled with a *WhereClause*, one is included as part of the common node specification. The final attribute, *Retrieve*, optimizes the context evaluation component by allowing the pattern table to not only carry the IIDs of objects, but also values needed to evaluate inter-class constraints in the WHERE clauses which involve classes stored at different sites.

The first specialization considered is the *LeafNode*. This represents the initial access to a set of class instances for the one class. A

WhereClause that consists of only intra-class constraints may restrict it.

In the *UnaryNode*, unary query operations may be applied to the results of child node. This too may have restrictions applied in a *WhereClause*. This *WhereClause* may refer to any of the classes in the *LeafNodes* that exist in this subtree.

The *BinaryNode* is similar to the *UnaryNode* except that now binary query operations can be applied to the results of two child nodes. Any inter-class restrictions that span both child nodes would normally be shown in the *WhereClause* attached to this node.

All the specializations of the QT Node shown so far are primarily for use in the context component of the QT. The final specialization, the *DoNode*, is for use in the operation component of the QT. This node specifies which method or set of methods is to be executed for the *PatternTable* associated with its child node. Any results from these executions are stored in the *TextResults* attribute for later presentation to the caller of the query.

Unary and binary query operations may be applied in the operation subtree, if the *PatternTable* is to be restricted further. This is useful if multiple subtrees are to be created in the operation subtree so that different sites execute methods on different components of the *PatternTable*.

Once the initial QT has been created, without any site data added, the query optimizer is applied to it. This optimizer determines where the operations are to be performed in the VE. This may involve manipulating the order in which the operations are performed in accordance with the association algebra, Su [19]. Once this task is completed, the optimized query plan must be executed.

Execution of the query plan as defined in the QT is carried out in two phases. In the first phase the context pattern is evaluated. This determines which objects are to have their methods executed in the second phase as defined in the DO component of the OQL query. The Distributed Context Subtree Handler (DCSH) performs the first phase using the QT returned from the query optimizer. The DCSH sends subqueries to participating legacy systems and coordinates these results with the results from the local OQL Engine that operates only on the local OOKBMS. This coordination may include further query evaluation based on the many subdatabase results being sent from the legacy systems.

Completion of the context component of the query by the DCSH results in a single subdatabase result represented by a *PatternTable* at the root node of the context subtree. The operations subtree in the Distributed Operations Subtree Handler (DOSH) references this node.

The DOSH issues remote execution requests to the legacy systems where the objects are stored for each of the objects identified in the result *PatternTable*. For this the *PatternTable* may be fragmented to reduce communication cost. Results from these executions are stored in the *TextResult* attributes and collected by the DOSH for presentation to the user at the completion of all operations.

2.3 Query Optimization Background

Query optimization has generated a great deal of interest in both Distributed Database Design (DDD) in relational systems by Apers [2][3], Ceri [4], Chu [7], Chen [6], Cornell [8], Hevner [11], Orlowski [13], Perrizo [14] and Yoo [27] as well as in deductive and object oriented databases Cheiney [5], Topaloglou [25] and Munoz [12]. Even though the emphasis of this paper is query optimization in a VE using an AOOKBMS to define it, the concepts used in relational DDD are transferable and hence deserve consideration and further discussion here. The objective here is to examine what factors influence query optimization and to take this into account when considering query optimization within the VE environment.

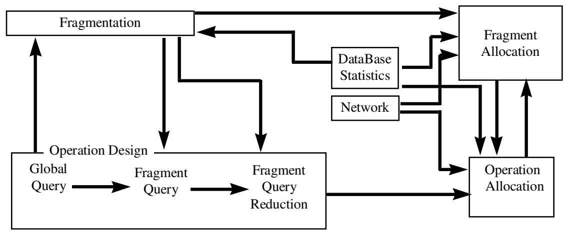

Figure 5-DDD Steps

The basic steps involved in DDD are shown in Figure 5 taken from Semeczko [17]. Query optimization may encompass part or all of the Operation Allocation (OA) and Operation Design (OD) steps from this diagram. In this diagram, the OD step consists of three steps. First, the global query, stated using the global schema. This is then transformed into a fragment query where the global references are converted into operations that recreate the global objects from fragments. In the last step, unnecessary fragments are eliminated. Within the VE environment fragments map to the data stored within the legacy systems and the global schema is the definition stored in the AOOKBMS.

The transformation of QTs for the purposes of optimizing relational processing is well covered by Chu [7]. Valid transformation rules for relational systems are also covered by Ceri [4]. As mentioned earlier, for the work here valid transformations are defined by the association algebra defined in Su [20].

These transformed QTs can then have network, fragment allocation and other statistical information applied to find allocations for the relational operations. This may also involve further transformations to the QTs as in Chu [7].

Many query optimization strategies do not cover all of these steps. Some discuss the optimization of queries on whole relations only and assume that their methods can be applied to fragments as well as in Apers [3], Hevner [11] and Yoo [27]. Others, such as Orlowski [13], assume that the queries are stated using the fragments. In Chen [6] the problem of carrying out a join between two global relations and its mapping to joins between the relevant fragments in an optimized manner is considered. So although the diagram above can be used as a guide, some problem spaces considered do vary the interconnection.

Another variation to the model shown relates to fragment allocation. Most of the works referenced above assume that the fragments are already allocated. In Cornell [8] an integer programming solution was used for the integrated problem of relation and process allocation. This problem was limited to relations with no fragmentation and no replication. Although we are dealing with legacy systems within the VE, fragment allocation can be accomplished here through migration of objects. This may also introduce replication.

For many of the query optimization strategies the QTs are simplified by only considering certain types of relational operations. In Orlowski [13] chain queries were assumed with the order of queries predetermined. Similarly, in Apers [3] and Hevner [11] the queries were limited to simple queries with the order determined by the size of the relations. This is unlike Chu [7] who considered a greater range of relational operations. Here, such considerations may be given to limiting the range of association operations considered. For example, only considering queries without "and" or "or" operations maybe a possibility.

Many of the query optimizing strategies do not consider general networks as Hevner [11] did, but assume that the network is simple as done by Chu [7], Chen [6], Apers [3] and Cornell [8]. That is, the costs between all pairs of sites are the same. This simplifies the cost functions used to determine the optimal query plans.

The cost functions that are used also vary depending on whether the objective is response time as used by Apers [3], Hevner [11] and Perrizo [14], total communication cost as in Chu [7], Orlowski [13] and Cornell [8] or some combination of both as in Chen [6]. In Ceri [4] it is argued that in geographically dispersed distributed systems, where the communication bandwidth is limited, considering only transmission costs is acceptable. It is also noted though, that this may not be the case in local area networks (LANs) where bandwidth is comparable to local I/O operations. Looking at these cost functions, there are three aspects to consider: environmental cost input, query cost input and selection data. Each is considered here.

First, environmental cost factors include both the network and the processors. These factors are:

- cost (transmission or delay) per unit of data transmitted on a communication link
- cost (transmission or delay) per unit of data processed on a processor

In both cases, the cost can be either an actual dollar cost or a delay. These costs may vary with time or may be assumed constant for a given period of time. Often these costs will be based on statistical data gathered over time and updated periodically. The volatility of these cost factors will influence the worth of any query optimization.

Second, for each query, the major cost factors relate to the size of the QT components. That is, the size of the fragments and any intermediate results. That is:

- size of initial fragments
- size of fragments after unary operations
- size of intermediate results

Here also, accuracy of such size calculations influences the worth of the query optimization process. Statistical data can be gathered following query execution to amend these calculations. In this project, the size of the *PatternTables* is one such consideration. How much of this data is stored and how frequently it is updated will also influence the query execution process and must be considered.

Finally, selection data relates to information necessary where a choice may be required. That is, where copies of resources, either data or hardware, are available a decision on which to use maybe needed. The decision may require the calculation of cost functions for different resource choices. This is covered by information regarding:

- location of fragments
- loads on communication links and processors
- reliability of communication links and processors

Based on the complexity of the assumptions being made with regard to operations, network topology and the cost factors included in the cost function, the results of any query optimization process are claimed to be either optimal as in Orlowski [13], Apers [3], Hevner [11] and Chu [7] or heuristic as in Apers [2] and Chen [6] or within some range of the optimal result as in Semeczko [17].

Given all these considerations the generation of distributed query plans in this project can now be discussed.

3. Query Plans

The objective in generating a query plan is usually to optimize the processing of a given query. To devise an optimization strategy, two criteria must be observed. First and foremost the query plan derived must be executable. Several factors determine whether such a plan is executable:

1. location of objects with respect to both method execution requests and initial context evaluations;
2. portability of method executable and possibly their source code;
3. which sites are able to perform certain query operations.

The second criteria is the efficiency of the execution plan. This is where the cost information discussed in Section 2.3 is taken into account in evaluating a cost function. This cost function is presented in Section 3.1. It is this second objective that most query optimization strategies concern themselves with while assuming that the first objective is always met.

The objective of this paper is to present a simple query optimization strategy that satisfies the first criteria while reducing the total communication cost from that of a centralized approach. This centralized approach involves migrating all the desired objects to the AOOKBMS site and then performing all operations and methods. While this centralized approach may return a result, it is very expensive, time consuming and relies upon the total portability of all objects to the AOOKBMS.

In achieving this objective, not only will it be shown that the NIIIP environment is an effective environment for distributed heterogeneous computations but that an active object oriented environment can be used to implement the query optimization process. Before presenting a query optimization strategy, the cost function that can be used to judge its effectiveness is presented.

3.1 Communication Cost Function

As mentioned in Section 2, many cost functions may be used depending upon what criteria the cost is to be evaluated on. In this paper the emphasis will be on the total communication cost. In addressing this problem, we assume that the controlling messages represent an insignificant cost when compared with the transmission of data. Hence, the cost functions developed concentrate only on the cost of data transmission and not control transmissions.

Total communication cost is based on the amount of data sent across the network in order to evaluate the query as defined by the QT. Based on the QT definition shown in Figure 4, three components that can be moved across the network are identifiable. These are: pattern tables representing the results of subqueries, objects being migrated between sites, and finally the movement of the final results to the query issuer. Given these components and the method of evaluation of a QT as described in Section 2.2 a cost function can be defined.

The cost function used to calculate the total communication cost for a query is defined by the following function:

$$Cost = \sum_{n_i}\left(\left(p_i^{'} + M_i^{'}\right) * c\left(n_i^{\#}, \rho\left(n_i\right)^{\#}\right)\right) + \sum_{n_i | t_i = do}\left(R_i^{'} * c\left(n_i^{\#}, Q^{\#}\right)\right) \tag{3.1}$$

Where:

$n_i \in N$ where N is the set of all nodes in the QT

$c(x, y)$ returns a positive number representing the cost of sending a unit of data from site x to site y

$n_i^{\#}$ site where the operation at node n_i is to be performed

$p_i^{'}$ size of pattern table p_i at node n_i

$\rho(n_i) \in N$ returns the parent node in the QT for node n_i

$t_i \in$ {Leaf,Binary,Unary,Do} returns the type of node that n_i is from the four available

$R_i^{'}$ returns the size of the result R_i evaluated from the Do node n_i

$Q^{\#}$ site where query was issued

$M_i^{'}$ size of migration component

In this function, the three earlier identified components are represented by two separate summations. The first summation is the pattern table and the migrated components and the second is the result component.

Given this cost function we can now present the strategy for determining the site allocations and migration specification for each node.

3.2 Simple Query Plan Generation

The strategy presented in this section makes no assumptions about the form of the query to be optimized. Because of this, no claims will be made about how efficient the generated plan will be. The only claim of this strategy is that it will generate a query plan that is valid and attempts to execute in a distributed manner. Future papers will concentrate on finding optimal query processing strategies for queries with given properties. The objective of this strategy however is to show that distributed query processing of a query in a distributed heterogeneous environment is possible.

The algorithm to be applied in this initial strategy assumes that each class is allocated to at most one site. Algorithm **SimplePlan** is described below:

ALGORITHM SimplePlan

INPUT: Query Tree Q,
Meta-Data in the AOOKBMS

OUTPUT: Query Tree Q

1. Initialize Q:

$$\forall n_i \bullet n_i \in N \wedge t_i = Leaf$$

$$n_i^{\#} = Class\ Site$$

$$\forall n_i \bullet n_i \in N \wedge t_i = Do$$

$$n_i^{\#} = Class\ Site\ for\ execution\ methods$$

2. Allocate all other Context Subtree Nodes

$$\forall n_i \bullet n_i \in N \wedge$$

$$t_i \in \{Binary, Unary, Leaf\} \wedge$$

$$n_i^{\#} \neq NULL$$

IF $\rho(n_i)^{\#} == NULL$ THEN

$$\rho(n_i)^{\#} = n_i^{\#}$$

ELSE IF $\rho(n_i)^{\#} \neq n_i^{\#}$ THEN

$$\rho(n_i)^{\#} = Q^{\#}$$

3. All remaining nodes processed at AOOKBMS

$$\forall n_i \bullet n_i \in N \wedge n_i^{\#} == NULL$$

$$n_i^{\#} = Q^{\#}$$

4. Return Q

In the algorithm **SimplePlan**, for Step 1 the LeafNode site allocations are set to the sites where their associated class is stored. Similarly the DoNodes have their sites set to the storage site of the class associated with the methods to be called[3]. After this, Step 2 proceeds up the QT allocating to the parent node of each Binary, Unary, or Leaf Node which has a site allocation. If this site has not been set before it will be that of the first child's node. If it has been set before, then it must be a binary node and hence will be either left as is or changed to the AOOKBMS site. It is left unchanged when the second child node has the same site setting and changed to the AOOKBMS if it does not. When all the parent nodes have been checked, the context evaluation component of the QT is completely allocated. All that remains are any nodes in the Do subtree. These may be performing operations on the *PatternTable* in order to allocate the execution requests to the various sites where the objects are stored. Such fragmentation operations will always be performed at the AOOKBMS site in order to reduce communication costs.

Once this has been completed the QT may be returned ready for execution. It may also be used as input to the cost function presented in Section 3.1 to evaluate the total communication cost of executing it.

[3] It is assumed here that the DoNodes are created with only those methods belonging to the same class being together.

Given this algorithm, we now present the NCL implementation of not only the **SimplePlan** algorithm, but also the mechanism to control its processing. This too is written in NCL.

4. Using NCL

The OSAM*.KBMS is not only an active objective oriented knowledge base system with a powerful object query language, OQL. To aid in the development of the methods in the class definitions a programming language has also been included. This programming language is called the NIIIP Common Language (NCL) as described in Su [22]. In this section we show how this language can be coupled with the OQL language and the active component of the OSAM*.KBMS to implement both the **SimplePlan** algorithm as well as the mechanism to execute it.

In Section 4.1 the execution mechanism is presented and in Section 4.2 the **SimplePlan** algorithm is presented. All NCL implementation code presented here applies to either the meta-model shown in Figure 2 or the QT definition shown in Figure 4.

4.1 Query Execution

Whichever query optimiser is used to produce a query execution plan, a mechanism must exist to control this execution. Current work in using OSAM*.KBMS to perform workflow execution would be a good candidate. As this ongoing work is not yet available and not associated directly with this project, an execution mechanism is to be created specifically for query execution.

The input to the query execution module is an *allocated* QT. In this system this is modelled by the *QueryTree* object which has the boolean flag *allocated* set to TRUE. This signifies that all the query tree nodes have an association with a site. The site associated is where the query optimizer has determined that the operation that the *QTNode* represents is to be performed. It is the role of the Query Execution module to process this query tree and coordinate the subqueries represented by the many subtrees in this query tree. This entails not only dispatching the subqueries to the nominated sites, but then coordinating the results and their integration via the remaining query operations to be performed at the AOOKBMS site.

In this first version of a distributed query processor, the following processing guidelines are followed:

- Subtrees allocated to the same remote site are sent to that remote site.
- Once migrated, execution of the subquery represented by this subtree can be requested.

- Once a remote subquery has been completed, its result is migrated to the AOOKBMS site for further processing.
- All nodes allocated to the AOOKBMS site are processed individually based on when the dependent query tree nodes are available. That is, when the child query tree nodes have been completed.

This first Query Execution model is defined as follows:

```
CLASS QueryExecution:
   QueryTree sqt;
   Site Home;        //Site where OOKBMS resides
                     //and final execution occurs
   Boolean finished;
METHODS:
   NodeExecute(QTNode qryop);
      // calls the query engine to carry out operation
      // remote and local
   SubQueryAllocate(QTNode: node);
      // Determine whether this node represents the
      // root of a subtree to be sent to a remote site
      // and execute
   SendSubTree(QTNode:node);
      // migrate the subtree with this node as its root
RULES:
   RULE Dependency
      // once an operation has completed - check if
      // any dependent operations can be executed
   TRIGGERRED AFTER
                NodeExecute(qryop: QTNode)
   ACTION
      // mark this operation as completed
      CONTEXT   q: QTNode
      WHERE   q = qryop
      DO     UPDATE(q, completed = TRUE)

      //If Parent is Unary then execute it - it is
      dependent
      CONTEXT   c: QTNode * p: UnaryNode
      WHERE   c = qryop AND p.completed = FALSE
      DO     NodeExecute(p)

         //If Parent is Binary and qryop is Left Child -
         //check both children completed
         CONTEXT   c: QTNode *>[LChild]
                   p: BinaryNode *<[RChild]
                   o: QTNode
         WHERE   c = qryop AND o.completed =
         TRUE
         DO     NodeExecute(p)

         //If Parent is Binary and qryop is Left Child -
         //check both children completed
         CONTEXT   c: QTNode *>[RChild]
                   p: BinaryNode *<[LChild]
                   o: QTNode
         WHERE   c = qryop AND o.completed =
         TRUE
         DO     NodeExecute(p)
   END;
END;

IMPLEMENTATION NodeExecute(QTNode: qryop)
   //Call query Engine to execute this Node if local
   IF QTNode.PerformAt = Home THEN
      LocalExec(QTNode);  //not defined yet -
                          //part of OQLEngine
ELSE //otherwise send an RPC
      RPC(QTNode);    //ask for its execution at the
                      //remote site
      migrate(QTNode); //get results back
END;

IMPLEMENTATION     SubQueryAllocate(QTNode:
node)
   IF node.PerformAt != Home THEN
      SendSubTree(node);
   ELSE {
      CONTEXT   p: QTNode * c: LeafNode
      WHERE   p = node AND c.Site = Home
      DO     NodeExecute(c)

      CONTEXT   p: QTNode OR
            {  *>[Child] c: QTNode,
               *>[Lchild] c:QTNode,
               *>[Rchild] c: QTNode}
      WHERE   p = node AND c.Site = Home
      DO     SubQueryAllocate(c)
      }
END;

IMPLEMENTATION SendSubTree(QTNode: node)
   migrate(QTNode, QTNode.PerformAt);
         //Migrate object and execute?
   CONTEXT   p: QTNode OR
            {  *>[Child] c: QTNode,
               *>[Lchild] c:QTNode,
               *>[Rchild] c: QTNode}
   WHERE   p = node
   DO     migrate(c, c.PerformAt)

   NodeExecute(node);
END;
```

4.2 SimplePlan Using NCL

In this section, we use NCL to implement the algorithm **SimplePlan** presented in the Section 3. The assumptions made for this implementation are as follows:

- Each class is stored at only one site.
- Unary operations performed on base classes (LeafNodes) are performed at their storage site.
- All other operations (represented by non-LeafNodes) shown in the QT are performed at either the site where the subtree results are stored (stored at the same site) or at the site where the OOKBMS is located.

This Query Optimizer is defined within the AOOKBMS as a class with the following definition:

```
CLASS QOP-Simple
   QueryTree * sqt;  //Pointer to the Query Tree being
                     //worked upon
   Site HM;          //Location of OOKBMS
METHODS:
   Init_Locations();  //Initialize the locations and sets
                      //LeafNodes "completed" flag to
                   TRUE
   Set_Location(qt: QTNode, opsite: Site);
```

```
            // Sets Site on QTnode to either Site or
            HM
   Complete();  //Sets all "completed" flags to FALSE
                //once all QTnodes are allocated a
                site

RULES:
   RULE setsites
   TRIGGERRED AFTER
         Set_Location(qt: QTNode, opsite: Site)
   CONDITION qt.completed = TRUE
   ACTION
     //Parent is unary
     CONTEXT   c:QTNode * p:UnaryNode
     WHERE     c = qt AND p.completed = FALSE
     DO        UPDATE(p, completed = TRUE),
               SetLocation(p, c.PerformAt)

     //Parent is binary and not set at all yet
     CONTEXT   c : QTNode * p : BinaryNode
               !>[PerformAt] Site
     WHERE     c = qt AND p.completed = FALSE
     DO        SetLocation(p, c.PerformAt);

     //Parent is binary, set, but not the same as second
     //child
     CONTEXT   c : QTNode * p : BinaryNode
               *>[PerformAt] Site
     WHERE     c = qt AND p.completed = FALSE
     AND
               c.PerformAt != p.PerformAt
     DO        UPDATE (q, completed = TRUE),
               SetLocation(q, HM);
     //Parent is binary, set and the same as second
     child
     CONTEXT   c : QTNode * p : BinaryNode
               *>[PerformAt] Site
     WHERE     c = qt AND p.completed = FALSE
     AND
               c.PerformAt != p.PerformAt
     DO        UPDATE (p, completed = TRUE);

     //No Parents - Last node - reset all flags
     CONTEXT   c: QTNode ! q : QTNode
     WHERE     c = qt
     DO        complete();
   END;
END;


IMPLEMENTATION Init_Locations():
   CONTEXT   Class * LeafNode
               *>[BelongTo] QueryTree
   WHERE    QueryTree = sqt
   DO       UPDATE (LeafNode,
               PerformAt = Class.class_stored);
END;

IMPLEMENTATION Complete(QTNode: node):
   CONTEXT   c: QTNode * p:QTNode
        // check if this goes both ways or only one
   WHERE      p = node
   DO         UPDATE (p, completed = FALSE),
              complete©
END;
```

This implementation of algorithm **SimplePlan** is different from that presented in Section 3.2 mainly because it is event driven rather than procedural. Its functionality is explained by the comments throughout the code.

5. Conclusions and Future Work

In this paper we have shown that the AOOKBMS is a suitable tool for the task of performing query optimization as well as controlling the execution of the finished query execution plan in a distributed heterogeneous VE. This was done by describing a simple algorithm for developing a distributed query execution plan and then showing its implementation in a programming language developed for our AOOKBMS. The implementation of the execution control was also presented along with a cost function for evaluating the total communication cost for processing the plan.

A distributed query processor with minimal query optimisation has been implemented at the University of Florida under the NIIIP project. Further work would involve the replacement of the query optimisation component by experimental query optimizers that will take into account certain query structures that allow for better optimizations. These query optimizers are currently under investigation and the subject of future papers.

Acknowledgment

Most of the research for this paper was carried out while George Semeczko was on Professional Development Leave from the Queensland University of Technology at the University of Florida. This work was supported by the Advanced Research Project Agency under ARPA Order #B761-00. It is part of the R&D effort of the NIIIP Consortium. The ideas and techniques presented here are those of the authors and do not necessarily represent the opinion of other NIIIP Consortium members.

References

[1] Alashqur A.M., Su S.Y.W., Lam H.X., *OQL: A Query Language for Manipulating Object-oriented Databases*, Proceedings of the Fifteenth International Conference on Very Large Data Bases, Amsterdam, 1989, pp433-442

[2] Apers P.M.G., "Data Allocation in Distributed Database Systems", *ACM Transactions on Database Systems*, vol.13, no.3, September 1988, pp 263-304

[3] Apers P.M.G., Hevner A.R., Yao S.B., "Optimization Algorithms for Distributed Queries", *IEEE Transactions on Software*

Engineering, vol.SE-9, no.1, pp. 57-68, January 1983

[4] Ceri S., Pelagatti G., "Correctness of Query Execution Strategies in Distributed Databases", *ACM Transactions on Database Systems*, vol.8, no.4, December 1983, pp577 - 607

[5] Cheiney J-P, Lanzelotte R.S.G., *A Model for Optimizing Deductive and Object-Oriented DB Requests,* Eighth International Conference on Data Engineering, Tempe, Arizona, February 1992

[6] Chen J.S. & Li V.O.K., "Optimizing Joins in Fragmented Database Systems on a Broadcast Local Network", *IEEE Transactions on Software Engineering*, vol.15, no.1, pp. 26-38, January 1989

[7] Chu W.W. & Hurley P., "Optimal Query Processing for Distributed Database Systems", IEEE Transactions on Computers, Vol C-31, No. 9, September 1982

[8] Cornell D.W. & Yu P.S., "On Optimal Site Assignment for Relations in the Distributed Database Environment",*IEEE Transactions on Software Engineering*, vol.15, no.8, pp. 1004-1009, August 1989

[9] Coulouris G., Dollimore J., Kindberg T., *Distributed Systems Concepts and Design*,2nd ed., Addison-Wesley Publishing Co., 1994

[10] Finin T., Weber J., Wiederhold G., et.al., *Draft Specification of the KQML Agent-Communication Language*, June, 1993, http://www.cs.umbc.edu/kqml/kqmlspec.ps,

[11] Hevner A.R. & Yao S.B., "Query Processing in Distributed Database Systems", *IEEE Transactions on Software Engineering*, Vol SE-5, No. 3, May 1979

[12] Munoz A., *An Extensible Query Optimizer Architecture for the TIGUKAT Objectbase Management System*, Technical Report TR 94-01, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, January 1994

[13] Orlowski M., *An Optimisation of Joins in Distributed Database System*, Technical Report 1/92, School of Information Systems, Faculty of Information Technology, Queensland University of Technology, January 1992

[14] [Perrizo89] Perrizo W., Lin J.Y.Y., Hoffman W., "Algorithms for Distributed Query Processing in Broadcast Local Area Networks", *IEEE Transactions on Knowledge and Data Engineering*, vol.1, no.2, June 1989, pp215-225

[15] Potharaju M., *Design and Implementation of a Query Processor For an Object-Oriented Database*, M.S. Thesis, Department of Computer and Information Sciences, University of Florida, 1995

[16] Semeczko G, Su S.Y.W., *Supporting Distributed Query Processing in a Heterogeneous,* Technical Report 96-032, Dept. of Computer and Information Science and Engineering, August, 1996, *http://www.cis.ufl.edu /research/tech-reports/tr96-abstracts.html*

[17] Semeczko G., *Design Issues in Distributed Databases for Wide Area Networks*, PhD Thesis, Department of Computer Science, University of Queensland, November, 1995

[18] Semeczko G., Su S.Y.W., *Supporting Object Migration in Distributed Systems,* Technical Report 96-029, Dept. of Computer and Information Science and Engineering, July, 1996, *http://www.cis.ufl.edu/research/tech-reports/tr96-abstracts.html*

[19] Su S.Y.W., Guo M., Lam H., *Association Algebra: A Mathematical Foundation for Object-Oriented Databases*, IEEE Transactions On Knowledge and Data Engineering, Vol. 5, No. 5, October 1993.

[20] Su S.Y.W., Krishnamurthy V., Lam H., *An Object-Oriented Semantic Association Model (OSAM*)*, Chapter 17 in Artificial Intelligence: Manufacturing Theory and Practice, ed Kumara S.T., Soyster A.L., Kashyap R.L., Institute of Industrial Engineers, Industrial Engineering and Management Press, Norcross, GA, 1989

[21] Su S.Y.W., Lam H., Arroyo-Figueroa J.,Yu T., Yang Z., *An Extensible Knowledge Base Management System for Supporting Rule-Based Interoperability Among Heterogeneous Systems*, 4th International Conference on Information and Knowledge Management, Maryland, USA, November 1995

[22] Su S.Y.W., Lam H., Yu T., Arroyo-Figueroa J., Yang Z., Lee S., *NCL: A Common Language for Achieving Rule-Based Interoperability among Heterogeneous Systems*, Journal of Intelligent Information Systems, Special Issue on Intelligent Integration of Information, Vol.6, 1996, pp. 171-198.

[23] Su S.Y.W., Lam H., Yu T., Lee S., Arroyo J., *On Bridging and Extending OMG/IDL and STEP/EXPRESS for Achieving Information Sharing and System*, EXPRESS Users' Group (EUG'95), Grenoble, France, October, 1995

[24] Su S.Y.W., Lam H.X., *An Object Oriented Knowledge Base Management for Supporting Advanced Applications*, 4th

International Hong Kong Computer Society Database Workshop, December 1992

[25] Topaloglou T., Illarramandi A., Sbattella L., *Query Optimization for KBMSs: Temporal, Syntactic and Semantic Transformations,* Eighth International Conference on Data Engineering, Tempe, Arizona, February 1992

[26] Wang, Fang, "The Design and Implementation of a Distributed Query Processor in a Heterogeneous Environment", Masters Thesis, Department of Computer and Information Science and Engineering, University of Florida, May 1997

[27] Yoo H., Lafortune S., "An Intelligent Search Method for Query Optimisation by Semijoins", *IEEE Transactions on Knowledge and Data Engineering*, vol.1, no.2, June 1989, pp226-237