Validating Mediator Cost Models with DISCO

Hubert Naacke[†] — **Anthony Tomasic**[⋆] — **Patrick Valduriez**[†]

[†] *INRIA Rocquencourt, 78153 Le Chesnay, France*
Hubert.Naacke@inria.fr, Patrick.Valduriez@inria.fr

[⋆] *Digital Integrity Inc., San Mateo, California*
tomasic@digital-integrity.com

ABSTRACT. DISCO *is a mediator system developed at INRIA for accessing heterogeneous data sources over the Internet. In* DISCO*, mediators accept queries from users, process them with respect to wrappers, and return answers. Wrapper provide access to underlying sources. To efficiently process queries, the mediator performs cost-based query optimization. In a heterogeneous distributed database, cost-estimate based query optimization is difficult to achieve because the underlying data sources do not export cost information.* DISCO*'s approach relies on combining a generic cost model with specific cost information exported by wrappers. In this paper, we propose a validation of* DISCO*'s cost model based on experimentation with real Web data sources. This validation shows the efficiency of our generic cost model as well as the efficiency of more specialized cost functions.*

KEY WORDS : *Heterogeneous Distributed Database, Query optimization, Cost model, Experimentation*

RÉSUMÉ. DISCO *est un système de médiation developpé à l'INRIA pour accéder à des sources de données hétérogènes réparties sur Internet. Dans* DISCO*, l'utilisateur pose des requêtes au médiateur. Le médiateur traite les requêtes en accédant aux données via les adaptateurs, et retourne la réponse à l'utilisateur. Pour être efficace le médiateur optimise les requêtes en se basant sur l'estimation de leur coût. L'estimation du coût est difficile car les sources hétérogènes n'exportent pas d'information de coût. La solution apportée avec* DISCO *consiste à combiner un modèle de coût générique avec des informations de coût exportées par les adaptateurs pour permettre au médiateur d'estimer le coût des requêtes hétérogènes. Dans cet article, nous proposons une validation du modèle de coût de* DISCO *par une expérimentation sur des sources de données réelles accessibles sur le Web. Cette validation montre l'efficacité de notre modèle de coût générique ainsi que l'efficacité de fonctions de coût plus spécialisées.*

MOTS-CLÉS : *Base de données distribuée hétérogène, Optimisation de requêtes, Modèle de coût, Expérimentation*

1 Introduction

The concept of a *mediator* [Wie93] has been proposed as a good basis for giving integrated views of multiple heterogeneous data sources. Declarative queries upon the views have to be processed efficiently by the mediator. The work described in this paper is part of the DISCO [TRV96, TRV98] project at INRIA. DISCO has a mediator based architecture for accessing heterogeneous distributed databases [ÖV99]. The architecture consists of *data sources* that provide raw data, *wrappers* that provide interfaces to data sources, *mediators* that provide declarative query access to multiple wrappers, and *clients* that provide queries to mediators and accept answers returned from mediators. Several projects follow a similar architecture (e.g. Garlic [C+95, RÖH99], DIOM [LP95], HERMES [SAB+97], COIN [B+97], IRO-DB [GFF97], MiroWeb [FGL+98]).

Declarative access in the form of queries on data sources gives a degree of freedom to the mediator to determine the best plan for the execution of the query. From a declarative query, the mediator can generate multiple access plans involving local operations at the data source level and global ones at the mediator level. The plans can differ widely in execution time due to varying local processing costs, communication costs, and mediator processing costs. The method for choosing the best plan remains an open issue. In classical database systems, the query *optimizer* generally implements a search strategy using a cost model. Plans are generated and compared using a cost estimate derived from database statistics and cost formulas to compute the cost of each operator of the plan. This approach cannot easily be applied to heterogeneous databases with multiple data sources because: (i) data sources do not report needed statistical information (e.g., HTML files, object-oriented databases); (ii) cost formulas for processing an operator (e.g., selection, or join) vary radically depending on the implementation of the wrapper and the underlying data source; (iii) communication costs are difficult to determine and may vary over time according to the network or system loads (e.g., on the Internet).

Various solutions to the cost estimate problem have been proposed in the past [BGW+81, AHY83, YC84]. Recently, the calibration approach was introduced in [DKS92] and extended to object systems in [GST96]. A calibrating procedure is proposed that estimates the coefficients of a generic cost model, which can be specialized for a class of systems. This approach has been implemented in Pegasus [SAD+95] and in the IRO-DB project [GGT95]. The main problem for calibration appears when a data source does not follow the generic cost model of these systems (which cannot be changed). We believe that this situation arises frequently in a heterogeneous environment. Another approach, proposed in the HERMES [ACPS96] project, records the cost information for every query issued to a data source. Cost estimates for new queries are based on the history of queries issued to a data source. Although very interesting for uniformly used sources, the approach is limited for data sources which are queried with dissimilar predicates or which are rarely queried.

DISCO's approach relies on combining a generic cost model with specific cost information exported by wrappers [NGT98]. The wrapper implementor specifies any part of the cost information of the data source, from nothing to everything. By default, the mediator implements its own generic cost model, and when possible, corrects it with the information imported from the wrappers. Thus, the generic cost model is used by the mediator for unknown data source operations, while the wrapper cost models provide, through a standard interface, more accurate cost formulas.

In this paper, now that the DISCO prototype has been implemented, we propose a validation of its cost model based on experimentation with real Web data sources. We prefer a real validation rather than simulation or benchmarking which have not yet been devised for mediator systems. The goal is to show that integrating the cost information of these data sources at the mediator level yields more efficient execution plans for practical queries.

The paper is organized as follows. Section 2 presents DISCO's architecture and generic cost model. to improve a previous generic multidatabase architecture. Section 3 describes how a wrapper provides the necessary statistics, size and cost computation rules. We provide a language for expressing this information. Section 4 presents our validation method. Section 5 describes the experimental system based on the DBLP and ACM Web data sources. Section 6 describes our performance experiments. Section 7 concludes.

2 Architecture and generic cost model

In this section, we introduce and discuss the original features of the architecture of DISCO [TRV98] with overview of the phases and steps required to process a query. We also introduce the generic cost model of the mediator.

2.1 DISCO architecture

The architecture of DISCO has been designed to avoid the drawbacks of heterogeneous distributed database systems (see Figure 1). First, the mediator does not use a uniform query language to communicate with local data source wrappers. Rather, low level algebraic operators are used and can be submitted to local sites according to capabilities information [KTV97]. Second, interaction between the wrapper and mediator occurs in two phases, the *registration* phase and the *query processing* phase. During the registration phase, mediators contact wrappers and check whether the export schema has been updated since last download. If yes, all the information required to use the wrapper is downloaded. The client dictionary is then updated. Having such dynamic export schema update procedures is an important feature for a system capable of handling a large number of data sources. Third, the mediator does not assume a uniform cost model for all data sources. Rather, it handles an extensible generic

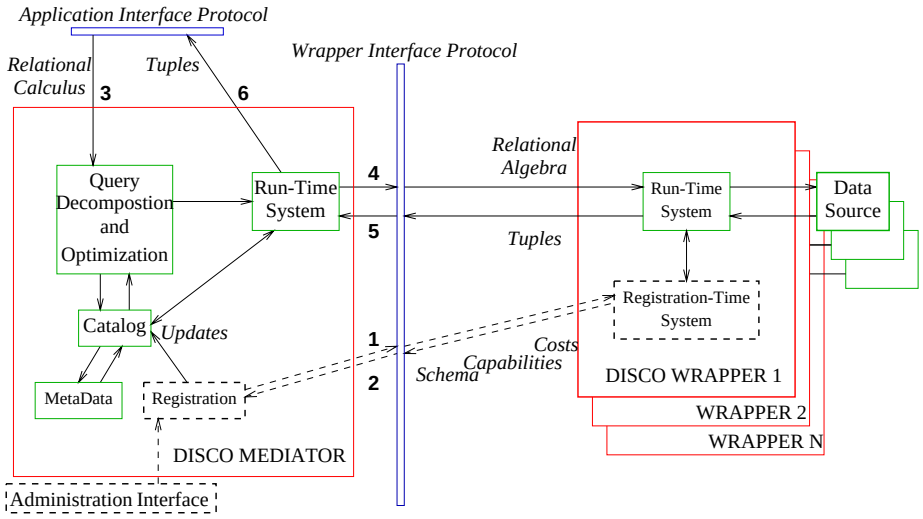

Figure 1: The DISCO system architecture.

cost model, which can be extended with specific cost information (i.e., rules and parameters) added to the export schema of each source by the local database administrator.

DISCO works as follows. During the registration phase Steps 1 and 2, the mediator (in response to instructions from the mediator administrator) calls the wrapper. The wrapper returns a collection of information needed for query processing. The collection contains the *schema* of the wrapper (reflecting the schemas of the underlying data sources, not shown here), *capabilities* of the wrapper (the set of operations the wrapper can execute), and *cost* information. Schema and cost information are stored in the mediator catalog. (The nature of the schema and capabilities information and its integration into query processing is considered elsewhere [KTV97]. In this paper we will assume that all wrappers can execute all operations.) We envision an administrative interface for both the mediator and wrapper to re-register wrappers. This interface is necessary when the cost formulas are improved by the wrapper implementor, or the statistics become out of date.

In the second phase, Steps 3 to 6, queries are processed. The second phase typically happens multiple times for every registration phase. In Step 3, the client issues a query to the mediator and waits for an answer. The mediator accepts the query and decomposes it into subqueries, one for each wrapper, and a composition subquery. In Step 4 the mediator issues the subqueries to the wrappers and waits for a response. The wrappers process the subqueries by consulting the associated data sources (not shown) and generate subanswers that are returned to the mediator in Step 5. The mediator combines the sub-

answers by using the composition subquery and generates the final answer that is returned to the client in Step 6. Note that in DISCO, the query in Step 3 is *declarative*, written in simple object/relational SQL language. The subqueries of Step 4 are *algebraic* and extend the relational logical operators.

To accomplish the translation from Step 3 to Step 4, the mediator does several things. It *parses* the client query, it *transforms* the query, written with respect to a global view, into a query over local schemas, and it *optimizes* the query to produce the best query execution *plan*. The mediator then executes the best plan, resulting in Step 4 and the subsequent steps. During optimization, the mediator estimates the cost of various operations and of entire plans. The mediator chooses the most specific information available as the result of registering wrappers. As discussed in the introduction, the best plan depends on good cost estimates for the subqueries sent to the wrapper. The mechanism described in this paper results in good cost estimates.

The mediator constructs several plans for the optimization of a query. A plan consists of a tree of algebraic operators. Although there exist many different data source managers, the basic algebraic operators are always the same; typically they include all operators of a classical object algebra. Thus, the mediator algebra covers the following common operators: (1) unary operators including `scan`, `select`, `project`, `sort`; (2) binary operators including `join`, `union`; (3) aggregate operators for elimination of duplicates or computing aggregate functions (e.g., `sum` and `average`); (4) an operator `submit` that is used to model the issuing of a subplan to a wrapper.

2.2 Mediator generic cost model

The role of the mediator optimizer is to select the most efficient plan among the alternatives based on the cost estimations. When no specific information are given by wrappers, the mediator estimates the cost of plans using a cost model. There are several major components of the cost : CPU cost, IO cost, and Communication cost.

The cost model depends on time parameters and statistical parameters. We assume in this paper a uniform communication cost; discrepancy of communication costs is a subject of future research. Time parameters come in three forms: the overhead required to start processing *TimeFirst*, the time required to deliver each tuple *TimeNext*, and the time to get all tuples *TotalTime*. *TimeFirst* accounts for query start up time and, in particular, `sort` operations. *TimeNext* gives the average time cost of each tuple. The time is measured in milliseconds.

For unary operators, the generic cost model of the mediator considers two cases: sequential scan, and index scan. The cost formulas are established using a calibrating approach [GST96]. These formulas requires the selectivity of a selection that can be derived from the minimum, maximum, and number of distinct values of the restricted attributes. Furthermore, to be able to select

the relevant formula, the data source must export the presence of indexes on attributes. In the generic cost model, clustering is not considered.

For binary operations, the generic cost model of the mediator considers three cases : nested loop join, hash join and dependant join. The dependant join uses the indexed access to the inner collection. The formulas are those of [GST96]. When an index is existing, the dependant join formula is selected, otherwise the best of the two others is chosen. Applying these formulas does not require more information that those define above for selections, as the join cardinality can be estimated as $1/Max(CountDistinct(A), CountDistinct(B))$. Thus, no further statistics are required for the generic cost model. The same is true for aggregate computation.

2.3 Implementation

The implementation of DISCO uses Java as the common language for mediator processing and wrapper processing. The cost formulas exported by wrappers are implemented as code generated from a compiler of the cost formula language. The resulting code is shipped to the mediator during the registration phase.

Encapsulating cost functions via code-shipping yields fast evaluation time for the functions during query optimization. Fast evaluation times are a requirement due to the computational intensity of query optimization. In addition, since cost formulas are shipped during the registration phase, the loading of cost formulas does not delay query processing. Finally, since the shipped code executes in the process space of the mediator, the entire library of code in the mediator (including the standard Java library) is available to the wrapper implementor when the cost formulas are defined.

3 Cost communication language

Local wrappers export data and operations described by the source administrator using a common object model. To base the system on solid foundations, we selected a subset of CORBA Interface Definition Language (IDL) [OMG95] to specify data source interfaces. To export statistics of collections, including cardinality, selectivity, object size, etc., we extend the *interface* body with a *cardinality* section. To overcome the limitations of using generic cost formulas in the mediator query optimizer, we also add a *cost formula* section which provides specific formula to the mediator. The *cost formula* section aims to better calculate the cost of an algebraic operation, i.e., a node in the query tree.

3.1 Exporting statistics

The local data sources also export statistics together with interfaces. Statistics are used as parameters in the mediator cost model formulas. Exported

statistics are simple, they describe data sources collections in the same way as in former calibrating approaches [DKS92, GST96]. The wrapper implementor expresses the statistical properties of a collection through two special methods attached to each interface description. To distinguish theses two methods from other possible ones, we add the keyword `cardinality` in front of the signatures of the both methods. The first method, named `extent` returns the number $CountObject$ of objects in the extent, the size $TotalSize$ of the extent in bytes, and the average size $ObjectSize$ of an object in bytes. The second method, named `attribute` describes, for a given attribute $AttributeName$, a boolean $Indexed$ indicating the existence of an index, the number $CountDistinct$ of distinct values for the attribute in the extent, and the minimum Min and maximum Max values for the attribute. Since the minimum and maximum values may be of various types, we encode this object in a special polymorphic `Constant` object. Figure 2 shows the two cardinality methods added into the interface definition of employee. The mediator calls the two methods extent and attribute during the interface registration, and stores the interface statistics in its catalog.

```
interface Employee {
 attribute Long salary;
 attribute String Name;
 cardinality
 extent(out long CountObject, out long TotalSize,
   out long ObjectSize);
 attribute(in String AttributeName,
   out Boolean Indexed,
   out Long CountDistinct,
   out Constant Min, out Constant Max); }
```

Figure 2: An example interface extended with statistics.

3.2 Exporting formulas

We assume that each data source wrapper is able to provide a basic object algebra. This is in general not true, but relaxing this assumption is out of the scope of this paper. However, even if implementing standard operators, a local data source may implement it in a very specific way, e.g., using a bit map index, a pointer chasing operator, or an efficient clustering algorithm. Thus, the generic cost model of the mediator optimizer will not be valid for this data source. To overcome this difficulty, we extend the interface definition with an optional new section to give cost formulas that will override the generic cost model of the mediator.

Cardinality of collection R	R.`CountObject`
Size of collection R in bytes	R.`TotalSize`
Average object size in R	R.`ObjectSize`
Index for attribute A	$R.A$.`Indexed`
Distinct values for attribute A	$R.A$.`CountDistinct`
Minimum value of attribute A	$R.A$.`Min`
Maximum value of attribute A	$R.A$.`Max`

Figure 3: Name scheme for statistics in a formula.

A *cost formula* is either collection oriented or operator oriented. If the cost formula is included inside the interface definition of a particular collection, it describes cost functions for operators on that particular collection. If the cost formula is apart from any interface definition, it describes cost functions that are not specially related to a particular collection but rather to an operator, i.e., the cost formula is valid for all collections of the source that have no collection specific cost formula. We will focus on the second kind of cost formulas; the first kind can be expressed using the same interface by naming explicitly the collection.

3.2.1 Cost formula syntax

Cost formulas have the standard mathematical syntax detailed in [NGT98]. Wrapper writers may use all the statistics, from the collection interfaces, by simply naming them. The naming convention is based on path expressions, such as *Collection.Attribute.Statistic*, where *Collection* is a collection name, *Attribute* is an attribute of the collection, and *Statistic* is a term referring to a statistic. *Attribute* and *Collection* may be omitted in non-ambiguous cases. Figure 3 lists the variable names for statistics that can be used in a formula.

3.2.2 Operator-formula attachment

In addition to writing formulas, the wrapper implementors indicate the operator on which a formula apply. We use a rule-based approach to bind each formula with its associated operator. We describe in the next section how the mediator matches such rules.

Each rule describes the cost for one operator. A rule is divided into a head and a body: (i) The rule head represents the operator and its arguments ; an argument may be bound to a collection or a predicate, or may be a free variable. (ii) The rule body is the formula itself ; the body may contain more than one formula depending on how many costs are provided by the wrapper implementor.

```
scan(employee) ⇐
   TotalTime    = 120 + Employee.TotalSize * 12

select(R, A = V ) ⇐
   CountObject  = R.CountObject * selectivity(A,V)
   TotalSize    = CountObject * R.ObjectSize
   TotalTime    = R.TotalTime + R.TotalSize * 25
```

Figure 4: Two example rules for computing both time and statistic formulas.

In Figure 4, we show some example rules for a `scan` operation on the collection employee and a `select` operation. In the figure, `employee` refers to the employee collection, A is a free variable that will be bound to a particular attribute name, V is a free variable that will be bound to a particular value, and `selectivity(`A`, `V`)` refers to an ad-hoc function defined by the wrapper implementor, that could handle, for example, histogram statistics [IP95, PIHS96]. Variables without a collection name refer to the result of the formula.

Given the plan `select(scan(employee), salary = 10)`, both of the rules match a part of the plan. The first rule matches `scan(employee)`, invoking the computation of `TotalTime` in the first rule. The second rule matches `select(c, salary = 10)`, where c represents the result of the `scan` and matches R, and A matches `salary` and V matches `10`, invoking the three computations of the formulas. The last computation uses the previous `TotalTime` result to compute a new `TotalTime` result. Note that for both rules, several formula are missing. Generic formulas (i.e., that of the generic cost model) are used in this case.

The rule approach provides a very large advantage to the wrapper implementor. The presence of free variables in the rule head makes very easy to adjust the cost precision by writing several rules, each rule more and more specific. However, the drawback to this expressiveness is the proliferation of query-specific cost rules that tends to slow down the cost estimate process. In other words the cost rules overriding mechanism should not induce significant workload on the mediator site. That is why we do not use the standard overriding mechanism of Java, but implement our own efficient one based on kind of virtual tables.

As we mentioned above, both statistical formulas and cost formulas are used to estimate the cost of a plan (tree). The cost of the execution of the plan is determined with a two step bottom-up algorithm. In the first step, each operator submitted to a remote data source is matched against the rule head patterns. If the operator name match the rule head, the binding mechanism unifies each variable in the pattern with a corresponding value from the operator being estimated. Therefore, two rules may have different matching levels: (i) unification on the collection name; (ii) unification on the attribute name; (iii)

unification on the predicate operation and the predicate arguments. In this case, we select the most specific rule, with more bound parameters. In case of multiple rules matching at the same level, we select the first one in the order given by the wrapper implementor.

4 Validation method

Our validation method is based on real experiments with Web data sources. For our experiments, we need to define several cost models with different precision so we can compare them. The precision of a cost model depends on the number of cost formulas, on the error rate in evaluating the formulas, and on the specialization level of the rules defining the formulas. We measure the efficiency of each cost model based on its precision.

The most efficient cost model is the one that yields the execution plan with the best response time. This is true if the plan having the best cost also has the best response time. In this case, we say that the efficiency of the cost model is 100%. However, this is not always true. This is false as soon as there is one plan whose cost is less than that of the plan having the best response time.

Let P be an execution plan, let $C(P)$ be its cost and let $T(P)$ be its response time. we measure the efficiency of a cost model as follows.

1. We evaluate the cost of the n plans produced by the optimizer. We order the plans by their cost and obtain the ordered set:

   $$\{P_1, \ldots, P_n \ \ such\ that\ C(P_i) < C(P_j)\}\ \forall\ 0 < i < j < n$$

   Plan P_1 of lowest cost is denoted by P_{Cmin}.

2. We run all execution plans and order them by their response time, yielding the ordered set:

   $$\{P'_1, \ldots, P'_n\ such\ that\ T(P'_i) < T(P'_j)\}\ \forall\ 0 < i < j < n$$

   Plan P'_1 with lowest response time is denoted by P_{Tmin}

3. we define efficiency E_{ff} as:

   $$E_{ff} = T(P_{Tmin})/T(P_{Cmin})$$

For our experiments, we choose rather simple queries which typically arise when accessing Internet data sources. Thus, the search space is small enough to be exhaustively explored and we can easily find the plan with best response time. The efficiency measured is then independent of the search strategy of the optimizer.

5 Experimental system

In this section, we describe our experimental system which is based on DISCO and two bibliographic data sources on the Web: DBLP and ACM. The experimental system is thus made of two wrappers and a mediator.

We chose real document data sources which are available on the Web and heterogeneous in terms of schema and processing capability. Unlike with a relational database which would accept all queries expressed on the schema in SQL, these data sources accept a limited subset of queries. Thus, we define a finite set of parametric queries representing all the queries which the source accepts.

We build the wrappers as follows. First, we analyze the content of the source to define the data with a relational schema. Second, we analyze the access methods supported by the data source. This yields the wrapper specification in terms of views on the source schema and of parametric queries on the views. These parametric queries represent the queries which the wrapper accepts from the mediator.

5.1 Wrapper for the ACM data source

The ACM Digital Library (URL: *www.acm.org/dl*) has three collections:

- Article(ref_article, title, conference, year)
- Writing(ref_article, author)
- Detail(ref_article, key_words, abstract, review)

We cannot access directly to collections Article, Writing, Detail because the source does not accept any query on these. However, the source contains a materialized view and accepts queries on this view. The materialized view V_{A1}(title, author, conference, year) is defined by the query:

V_{A1}: **select** `title, author, conference, year`
from `Article a, Writing e`
where `a.ref_article = e.ref_article`

The queries accepted by the wrapper on the view V_{A1} have the form:

$Q_{A1}(X)$: **select** `*` **from** V_{A1}
where $pred(X)$

The select predicate $pred(X)$ of $Q_{A1}(X)$ must contain the indexed attribute `author`. It may also contain the other attributes `conference` and `year`. The grammar of the select predicate based on X = (x_1, x_2, x_3) is:

<pred(X)> := <compulsory_pred(X)> (**and** <optional_pred(X)>)*
<compulsory_pred(X)> := author = x_1
<optional_pred(X)> := conference = x_2 | year = x_3

Let $\{Q_{A1.i}(X)\}\ \forall\ i \in [1,n]$ be the set of queries $Q_{A1.i}(X)$ whose predicate $pred(X)$ respects this grammar. The set is finite ($n = 4$). The four queries are:

$Q_{A1.1}(X)$: `select(author=`x_1`,` V_{A1}`)`

$Q_{A1.2}(X)$: `select(author=`x_1 `and conf=`x_2`,` V_{A1}`)`

$Q_{A1.3}(X)$: `select(author=`x_1 `and year=`x_3`,` V_{A1}`)`

$Q_{A1.4}(X)$: `select(author=`x_1 `and conf=`x_2 `and year=`x_3`,` V_{A1}`)`

5.2 Wrapper for the DBLP data source

The Digital Bibliography & Library Project (URL: *dblp.uni-trier.de*) contains four collections:

- Publication(ref_article, title, ref_conf)
- Writing(re_article, ref_author)
- Author(ref_author, name, home_page)
- Conference(ref_conf, name, year, location)

The first materialized view V_{B1}(title, author, conference, year) is defined by the query:

V_{B1}:
```
select p.title, a.name as author,
   c.name as conference, c.year
from Publication p, Writing e, Author a, Conference c
where p.ref_article = e.ref_article
   and p.ref_conf = c.ref_conf
   and e.ref_author = a.ref_author
order by title
```

The queries accepted on the view V_{B1} are select operations on `author` or `title`:

$Q_{B1}(x)$:
```
select * from V_B1
where author = x
```

$Q_{B2}(x)$:
```
select * from V_B1
where title = x
```

We can also obtain the list of all authors:

Q_{B3}: **select** `name` **from** `Author`

The other materialized views are:

- V_{B2}(conference, ref_conf)
- V_{B3}(ref_conf, year, ref_article)
- V_{B4}(ref_article, title, author).

They are defined by the queries:

V_{B2}:
```
select distinct conference, f(conference) as ref_conf
from V_B1
order by conference
```

V_{B3}:
```
select distinct t.ref_conf, year,
    f(conference, year) as ref_year
from V_B1 a, V_B2 t
where t.conference = a.conference
order by year
```

V_{B4}:
```
select distinct t.ref_year, title, author
from V_B1a , V_B3 t
where t.year = a.year
order by title
```

The queries accepted by the wrapper on these views are:

Q_{B4}:
```
select conference, ref_conf
from V_B2
```

$Q_{B5}(x)$:
```
select year, ref_year
from V_B3
where ref_conf = x
```

$Q_{B6}(x)$:
```
select title, author
from V_B4
where ref_year = x
```

5.3 Mediator for the bibliographic data sources

The integrated schema is a set of views on the schemas exported by the data sources. We define the integrated view Pub(title, author, conference, year) that is the union of views exported by the adapters.

Pub:

```
        select * from V_A1
    union
        select * from V_B1
    union
        select title, author, conference, year
        from V_B2 v1, V_B3 v2, V_B4 v3
        where v1.ref_conf = v2.ref_conf
            and v2.ref_year = v3.ref_year
```

At the mediator level, we can observe that Pub is the union of three queries that are semantically equivalent. The three queries produce the same result. This redundancy is used by the optimizer to widen the search space.

6 Performance experiments

The main goal of our performance experiments is to show that a specialized cost model is more efficient than the mediator generic cost model. In this section, we describe the query and execution plans for the experiments and the cost models of the wrappers and of the mediator. Then we assess the efficiency of the cost model.

6.1 Query and execution plans

Query Q_1 is a select on three attributes: "what are all publications written in journal x_1 since year x_2 by author x_3 ?"

$Q_1(x_1, x_2, x_3)$:

```
select *
from Pub
where conference = x1
   and year > x2
   and author = x3
```

Query Q_1 is of the general form:

$Q(x_1, x_2, \ldots, x_n)$:

```
    select * from R
    where R.a1 op x1
       and R.a2 op x2
       and ... and R.an op xn
with op ∈ {=, <, >}
```

The optimizer decomposes the select query in one logical operator with n select predicates. Then, the optimizer generates execution plans by distributing

selections between the mediator and the wrappers. As all the wrappers may be able to process the selections, the optimizer generates all the possible distribution of for each wrapper. Furthermore, the wrappers may be able to process a selection on more than one predicate. Then a select operator can be a conjunct of several predicates.

Let m be the number of wrappers, the distinct number of plans is :

$$m * (C_0^n + C_1^n + \ldots + C_n^n) = m * 2^n$$

The select predicates of Q_1 are on attributes `conference`, `year`, and `author`. The optimizer generates 16 plans. The plans $P_{A1}, \ldots, P_{A8}$ are accessing the ACM wrapper. The plans $P_{B1}, \ldots, P_{B8}$ are accessing the DBLP wrapper. The distribution of select operations for plans P_{A1} to P_{A8} is shown in Figure 5.The predicates marked with **m** are processed by the mediator, those marked with **w** are processed by the wrappers. The distribution of select operations for plans P_{B1} to P_{B8} is respectively the same as for plans P_{A1} to P_{A8}.

$\downarrow$ Predicate | Plan $\rightarrow$	P_{A1}	P_{A2}	P_{A3}	P_{A4}	P_{A5}	P_{A6}	P_{A7}	P_{A8}
conference $= x_2$	w	m	w	w	m	m	w	m
year $> x_3$	w	w	w	w	m	w	m	m
author $= x_1$	w	w	w	m	w	m	m	m

Figure 5: Equivalent plans for Q_1

The wrappers do not have the capabilities to process all these plans. The optimizer eliminates plans P_{A4}, P_{A6}, P_{A7}, P_{B4}, P_{B6} and P_{B7} which require the wrappers to perform a select without attribute `author` which is compulsory. The optimizer also eliminates plans P_{B1}, P_{B2}, P_{B3} because DBLP wrapper can only perform a selection on the `author` attribute. The remaining plans are P_{A1}, P_{A2}, P_{A3}, P_{A5}, P_{B5} and P_{B8}.

For each plan we detail the work performed by the wrappers. To process plans P_{A1}, P_{A2}, P_{A3}, P_{A5}, we submit the queries $Q_{A1.i}(pred, X)$ to the ACM wrapper. To process plan P_{B5}, we submit the queries $Q_{B1}(X)$ to the DBLP wrapper. To process plan P_8, we submit Q_{B4} to the DBLP wrapper and perform two dependent joins by submitting $Q_{B5}(X)$ and $Q_{B6}(X)$ to the wrapper. Each query submitted to a wrapper is then filtered by a select at the mediator level.

`Let X =` (x_1, x_2, x_3)
$P_{A1}(X)$: `submit(`$Q_{A1.4}(X)$`)`

$P_{A2}(X)$: `select(conf=`x_2`, submit(`$Q_{A1.3}(X)$`))`

$P_{A3}(X)$: `select(year>`x_3`, submit(`$Q_{A1.2}(X)$`))`

$P_{A5}(X)$: select(conf=x_2, year>x_3, submit($Q_{A1.1}(X)$))

$P_{B5}(X)$: select(conf=x_2, year>x_3, submit($Q_{B1}(X)$))

$P_{B8}(X)$: select(author=x_1,
join(select(year>x_3,
join(select(conf=x_2, submit(Q_{B4})),
submit(Q_{B5}(ref_conf)))),
submit(Q_{B6}(ref_year))))

6.2 Wrapper cost model

We first propose a generic cost model which should be simple and efficient. Then we extend the generic cost model for the DBLP wrapper with specific formulas.

The generic cost model defines the cost of a select and of a project. The select cost function takes into account the indexed access on attribute `author`. The other attributes are not indexed. Let $||R||$ be the cardinality of the collection R and $|R|$ be the size of R (i.e., $||R|| = R.CountObject$, $|R| = R.TotalSize$), the generic cost model is :

MC_1 :

$C(select(Pred, R))$	$TotalSize = \|R\| * selectivity(Pred)$
	$TotalTime = IO * \|R\| + CPU * \|\|R\|\|$
$C(select(author = X, R))$	$TotalTime = IO * TotalSize$
$C(proj(att, R))$	$TotalSize = \|\|R\|\| * \|att\|$
	$TotalTime = IO * \|R\| + CPU * \|\|R\|\|$

The selectivity of a simple predicate is fixed. The selectivity of a conjunct of predicates is the product of the predicate selectivities. The generic cost model is the same for the two wrappers. The coefficents of the generic formulas are calibrated for each wrapper.

We observe that the efficiency of the cost model decreases as the query selectivity increases. Thus, we complement this cost model with specific formulas. The cost model we obtain has 100% efficiency independent of query selectivity. It takes into account materialization and contains cost functions associated with queries Q_{B4}, Q_{B5} and Q_{B6}.

MC_2 :

MC_1 +	
$C(Q_{B4})$	$TotalSize = D_{conference}$
	$TotalTime = TotalSize * IO;$
$C(Q_{B5})$	$TotalSize = D_{year}$
	$TotalTime = TotalSize * IO;$
$C(Q_{B6})$	$TotalSize = \|V_{B4}\|/(D_{conference} * D_{year})$

$$TotalTime = TotalSize * IO;$$

with $D_{conference} = 80, D_{year} = 14$

The distribution of attributes `conference` and `year` is uniform. Attributes `conference`, `year` and `author` are independent. Therefore, the cost of queries Q_{B5} and Q_{B6} is proportional to the selectivity of predicates `conference=`x_2 and `year>`x_3 respectively.

6.3 Cost model of the mediator

The operations processed by the mediator are select, project, hash join and dependent join. The cost function for communication depends on the minimal bandwidth between the mediator and the wrappers. The cost of a select depends on the predicate selectivity. The selectivity of an equality predicate is :

$\mathsf{selectivity}(att = x) = 1/D_{att}$

The selectivity of the other predicates is:

$\mathsf{selectivity}(att < x) = 1/((x - min)/(max - min))$
$\mathsf{selectivity}(att > x) = 1/((max - x)/(max - min))$

The mediator cost model is:

$C(select(att\ op\ X, R))$
  $CountObject = ||R|| * selectivity(att\ op\ X)$
  $TotalTime = IO * |R| + CPU * ||R||$
$C(depjoin(R_1, R_2, r_1.x = r_2.y))$
  $TotalTime = R_1.TotalTime + ||R_1|| *$
  $(submit(select(R_2, r_2.y = 0))).TotalTime$
$C(submit(Q))$
  $TotalTime = COM * |Q|$

6.4 Efficiency of the cost model

The plan used for execution is chosen among the 6 plans. It is the lowest cost plan. The first optimization uses the generic cost model only. Plan P_{A1} evaluates lower than all other plans whatever is the selectivity. Plan P_{A1} which is chosen pushes the three select operations on the wrapper. The second optimization uses a more specialized cost model with cost functions associated with Q_{B4}, Q_{B5}, and Q_{B6}. The plan with best cost is P_{A1} or P_{B8} depending on the predicate selectivity. We compute the value of the selectivity when the costs of P_{A1} and P_{B8} are equal, this is for a selectivity of 0.008.

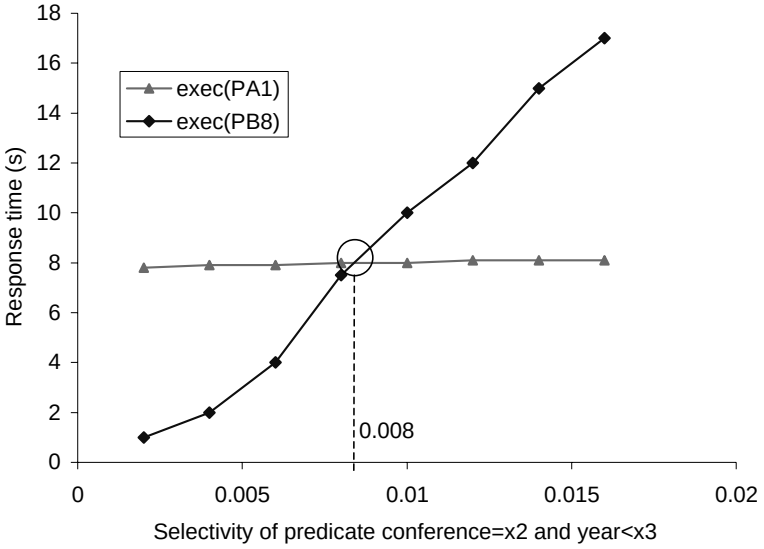

Figure 6: Response time of plans P_{A1} and P_{B8}

We measure the response time of plans P_{A1} and P_{B8} when the selectivity of predicate `year<X` is varying. Figure 6 shows the response times of plans P_{A1} and P_{B8} according to the selectivity of select on attributes `conference` and `year`.

We observe that the value of the selectivity when P_{A1} and P_{B8} have equal execution time is also 0.008. The selectivity is the same for the point of equal cost and for the point of equal response time. This is because the generic cost model as been accurately calibrated.

With high selectivities (> 0.008), it is better to perform all select operations on the wrapper. Both cost models lead to this solution. Thus, the generic cost model is as efficient as the specialized one. For lower selectivity (< 0.008), it is faster to perform only the two select operations (conference, year) on the wrapper. In this case, the specialized cost model has 100% efficiency whereas the generic cost model efficiency is decreasing. To analyze more precisely this loss of efficiency, we compute the efficiency of the generic cost model. This is the response time ratio of the two following plans : (i) the plan chosen by the specialized cost model, that is P_{A1} for selectivity < 0.008 and P_{B8} for selectivity > 0.008; (ii) the plan P_{A1} chosen by the generic cost model for any selectivity. Figure 7 shows the efficiency of the two cost models according to the selectivity of select on attributes `conference` and `year`. For low selectivity (x2 is set to the lower bound of the year domain) the generic cost model as a minimal efficiency of 12%.

We now vary the selectivity of predicate `author =` x_1. The response time of plans P_{A1} and P_{B8} is shown in Figure 8.

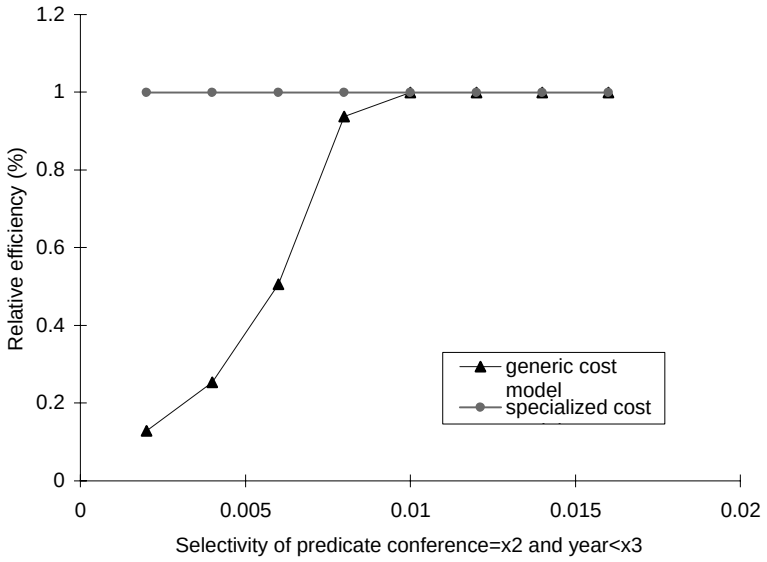

Figure 7: Relative efficiency of the cost models

With low selectivity (< 0.006), the response time of plan P_{A1} is less than that of plan P_{B8}. It is faster to perform all select operations on the wrapper. The generic cost model has 100% efficiency. However, for higher selectivities (> 0.006), it is better to perform only two select operations on the wrapper. Figure 9 shows that the efficiency of the generic cost model drops as the selectivity increases. Thus, we have the inverse result of the previous experiment. This is because when the selectiviy of predicate `author=`x_1 increases, the size of the result of accessing the index `author=`x_1 increases. However, for plan P_{B8}, the execution time of queries Q_{B4}, Q_{B5} and Q_{B6} remains the same since predicates `conference=`x_2 and `year>`x_3 have constant selectivity.

From these two complementary experiments, we can conclude that the generic cost model has 100% efficiency only on the interval [0.006, 0.008]. Outside this interval, its efficiency decreases. The specialized cost model has 100% efficiency in all experiments. To maintain such 100% efficiency, a sufficient condition is that the point at which the plans have equal cost and the point at which the plans have equal response time are the same. This is a transition point that splits the set of experimented queries in two, according to the predicate selectivity. This condition is verified for our specialized cost model since the response times of plans P_{A1} and P_{B8} are linear versus selectivity and fit our model.

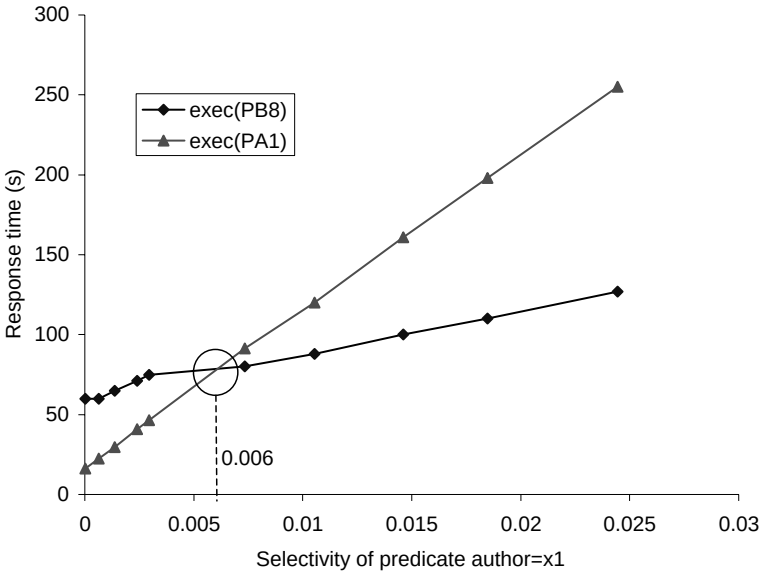

Figure 8: Response time of plans P_{A1} and P_{B8}

7 Conclusion

The DISCO project has developed a research prototype of components for searching and integrating information over distributed heterogeneous data sources. The target applications of this project are those of Internet and Intranet which typically require integration of a large number of diverse data sources. Since each data source generally performs operations in a unique way, the cost for performing an operation may vary a lot from one wrapper to another. DISCO addresses this heterogeneous cost model problem through an extensible cost model integrated within the mediator component.

In this paper, we have described a validation of this cost model based on experimentation with Web bibliographic data sources. Our experimental system consists of two wrappers for the DBLP and ACM data sources and of a mediator. Such system provides various access methods and materialized views which yields various execution plans, even for simple queries. Using our cost communication language, we could precisely define the heterogeneous aspects of the cost model through rules.

Our experiments have measured the efficiency of our model for choosing between two access methods. With linear modeling of access methods, we can choose the best plan for any query selectivity except when plans have close response times. We showed that the generic cost model has 100% efficiency in a restricted interval corresponding to low selectivity queries, for a restricted set

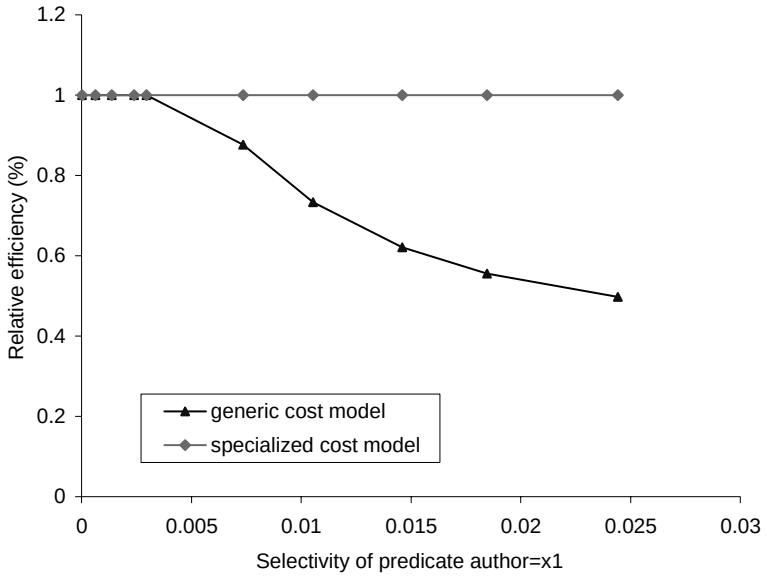

Figure 9: Efficiency of the generic cost model versus the selectivity of author=x_1

of queries. Outside this interval, its efficiency decreases significantly and a more specialized cost model is necessary.

In addition to generic formulas, the specialized cost model contains three specialized cost formulas for accessing materialized views. Theses formulas are easily calibrated and do not require much work for the wrapper implementor. However they are sufficient to yield 100% efficiency of the mediator cost model.

More complete and detailed experimentation can be found in [Naa99]. In particular it shows that our cost model is a reasonable solution for optimization of inter-site joins between large data sources in a real hospital application.

Acknowledgments

The authors wish to thank Georges Gardarin for his contribution in the definition of DISCO's cost model.

References

[ACPS96] S. Adali, K. Candan, Y. Papakonstantinou, and V. Subrahmanian. Query Caching and Optimization in Distributed Mediator Systems. In *ACM SIGMOD Int'l Conf. on Management of Data*, Montreal, Canada, 1996.

[AHY83] P. Apers, A. Hevner, and S. Yao. Optimization Algorithms for Distributed Queries. *IEEE Transactions on Software Engineering*, 9(1), 1983.

[B+97] S. Bressan et al. The Context Interchange Mediator Prototype. In *ACM SIGMOD Int'l Conf. on Management of Data*, Tucson, Arizona, 1997.

[BGW+81] P. Bernstein, N. Goodman, E. Wong, Christopher Reeve, and J. Rothnie. Query Processing in a System for Distributed Databases (SDD-1). *ACM Transactions on Database Systems (TODS)*, 6(1), 1981.

[C+95] M. Carey et al. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. In *Fifth Int'l Workshop on Research Issues in Data Engineering - Distributed Object Management (RIDE-DOM)*, Taipei, Taiwan, 1995.

[DKS92] W. Du, R. Krishnamurthy, and M.-C. Shan. Query Optimization in a Heterogeneous DBMS. In *18th Int'l Conf. on Very Large Data Bases (VLDB)*, Vancouver, Canada, 1992.

[FGL+98] P. Fankhauser, G. Gardarin, M. Lopez, J. Muñoz, and Anthony Tomasic. Experiences in Federated Databases: From IRO-DB to MIRO-Web. In *24th Int'l Conf. on Very Large Data Bases (VLDB)*, New York City, USA, 1998.

[GFF97] G. Gardarin, B. Finance, and P. Fankhauser. Federating Object-Oriented and Relationnal Databases: The IRO-DB Experience. In *2nd IFCIS Intl. Conf. on Cooperative Information Systems (CoopIS)*, Kiawah Island, South Carolina, 1997.

[GGT95] Georges Gardarin, J.-R. Gruser, and Z.-H. Tang. A Cost Model for Clustered Object-Oriented Databases. In *21st Int'l Conf. on Very Large Data Bases (VLDB)*, Zurich, Switzerland, 1995.

[GST96] G. Gardarin, F. Sha, and Z.-H. Tang. Calibrating the Query Optimizer Cost Model of IRO-DB, an Object-Oriented Federated Database System. In *22nd Int'l Conf. on Very Large Data Bases (VLDB)*, Mumbai (Bombay), India, 1996.

[IP95] Y. Ioannidis and V. Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. In *ACM SIGMOD Int'l Conf. on Management of Data*, San Jose, Calif., 1995.

[KTV97] O. Kapitskaia, A. Tomasic, and P. Valduriez. Dealing with Discrepancies in Wrapper Functionality. In *13ème journées de Bases de Données Avancées*, Grenoble, France, 1997.

[LP95] L. Liu and C. Pu. Distributed Interoperable Object Model and Its Application to Large-scale Interoperable Database Systems. In *4th Int'l Conf. on Information and Knowledge Management (CIKM)*, Baltimore, Maryland, 1995.

[Naa99] Hubert Naacke. *Modèles de coût pour médiateurs de bases de données hétérogènes.* Thèse de doctorat, Université de Versailles-Saint Quentin, 1999.

[NGT98] H. Naacke, G. Gardarin, and A. Tomasic. Leveraging Mediator Cost Models with Heterogeneous Data Sources. In *14th Int'l Conf. on Data Engineering*, Orlando, Florida, 1998.

[OMG95] Object Management Group OMG. *The Common Object Request Broker: Architecture and Specification Revision 2.0.* Object Management Group, Framingham, MA, 1995.

[ÖV99] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, 2nd Edition.* Prentice Hall, 1999.

[PIHS96] Viswanath Poosala, Yannis E. Ioannidis, Peter J. Haas, and Eugene J. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. In *ACM SIGMOD Int'l Conf. on Management of Data*, Montreal, Canada, 1996.

[RÖH99] M. Roth, F. Özcan, and L. Haas. Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System. In *25th Int'l Conf. on Very Large Data Bases (VLDB)*, Edinburgh, Scotland, 1999.

[SAB+97] V. S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. J. Lu, A. Rajput, T. J. Rogers, R. Ross, and C. Ward. HERMES: Heterogeneous Reasoning and Mediator System. submitted for publication, 1997.

[SAD+95] M. Shan, R. Ahmen, J. Davis, W. Du, and K. William. Pegasus: A Heterogeneous Information Management System. In *Modern Database Systems: The Object Model, Interoperability, and Beyond.* ACM Press, 1995.

[TRV96] A. Tomasic, L. Raschid, and P. Valduriez. Scaling Heterogeneous Database and the Design of DISCO. In *16th Int'l Conf. on Distributed Computing Systems (ICDCS)*, Hong Kong, 1996.

[TRV98] A. Tomasic, L. Raschid, and P. Valduriez. Scaling Access to Heterogeneous Data Sources with DISCO. *IEEE Transactions on Knowledge and Data Engineering*, 10(5), 1998.

[Wie93] G. Wiederhold. Intelligent Integration of Information. In *ACM SIGMOD Int'l Conf. on Management of Data*, Washington, D.C., 1993.

[YC84] C. T. Yu and C. C. Chang. Distributed Query Processing. *ACM Computing Surveys*, 16(4), 1984.