UNIVERSITE TOULOUSE III – PAUL SABATIER
UFR MIG, Laboratoire IRIT

THESE

en vue de l'obtention du

DOCTORAT DE L'UNIVERSITE DE TOULOUSE
délivré par l'Université Toulouse III – Paul Sabatier
Discipline : Informatique

présentée et soutenue

par

Belgin ERGENÇ

le 24 janvier 2008

Titre :

Un modèle d'exécution de requêtes mobiles pour des sources à accès restreints en environnement d'intégration de données

JURY

Bruno DEFUDE	Professeur à l'Institut National des Télécommunications (Rapporteur)
Abdelkader HAMEURLAIN	Professeur à l'Université Paul Sabatier (Directeur de thèse)
Serge MIRANDA	Professeur à l'Université de Nice Sophia Antipolis (UNSA)
Franck MORVAN	Maître de Conférences – HDR à l'Université Paul Sabatier
Halis PUSKULCU	Professeur à l'Institut des Technologies d'Izmir
Mesut RAZBONYALI	Professeur à l'Université Maltepe - Istanbul (Rapporteur)
Florence SEDES	Professeur à l'Université Paul Sabatier
Halil SENGONCA	Professeur à l'Université Egé - Izmir (Codirecteur de thèse)

Abstract: Query optimization in data integration systems over large scale network, faces the challenges of dealing with autonomous, heterogeneous and distributed data sources, dynamic execution environment and changing user requirements. These issues initiate the need for crafting the traditional optimization methods in a way to produce stable query execution plans, use execution models which are able to adapt to run-time conditions and handle source restrictions. Centralization of the control in adaptive optimization methods result in bottleneck due to large amounts of message passing towards to the site of the central authority over large scale network where network bandwidth is low and network latency is high. In order to overcome this obstacle adaptive query optimization requires decentralized methods. A mobile execution model with mobile relational operators that are able to adapt in an autonomous way and focus on reducing of transfer cost is worth considering in large scale distributed data integration environment. However, this mobile query execution model needs to be extended with new operators to handle source restrictions. In this perspective we propose mobile relational operators developed for restricted sources. Another proposition is related with initial placement of mobile relational operators. The challenge is on defining a placement which would allow acceptable performance instead of a good placement which would cause dramatic performance sometimes at run-time. Finally we present and analyze a performance evaluation on the methods proposed.

Keywords: Data integration, Distribution in large scale, Adaptive optimization, Mobile execution model, Restricted sources, Initial placement, Performance evaluation.

Résumé : L'optimisation de requêtes dans les systèmes d'intégration de données réparties sur un réseau à grande échelle pose des problèmes liés à l'autonomie, l'hétérogénéité et la distribution des sources de données, l'environnement d'exécution dynamique et aux besoins changeant des utilisateurs. Résoudre ces problèmes nécessite de revisiter les méthodes d'optimisation traditionnelles afin de permettre la génération de plans d'exécution stables et d'utiliser des modèles d'exécution capables de s'adapter aux conditions d'exécution et de tenir compte des sources à accès restreints. La centralisation du contrôle dans les méthodes d'optimisation dynamique engendre un goulot d'étranglement dû au nombre important de messages transmis au site contrôlant l'optimisation à travers un réseau à grande échelle (à faible bande passante et à forte latence). Afin de résoudre ce problème, l'optimisation dynamique des requêtes nécessite de décentraliser ces méthodes. Dans un environnement d'intégration de données distribuées à grande échelle, un modèle d'exécution mobile, avec des opérateurs relationnels mobiles capables de s'adapter de façon autonomes et réduisant les coûts de communication a été proposé dans l'équipe. Cependant, ce modèle d'exécution de requêtes mobiles nécessite d'être étendu avec des nouveaux opérateurs afin de tenir compte des sources à accès restreints. Dans cette perspective, nous proposons des opérateurs relationnels mobiles développés pour des sources à accès restreints. Une seconde proposition est liée au placement initial des opérateurs mobiles. Le but est de déterminer un placement permettant d'obtenir des performances acceptables plutôt qu'un bon placement engendrant parfois de très mauvaises performances lors de l'exécution. Enfin, nous évaluons les performances des méthodes proposées.

Mot-Clés : Intégration de données, Répartition à grande échelle, Optimisation dynamique de requêtes, Modèle d'exécution de requêtes mobiles, Sources à accès restreints, Placement initial, Evaluation des performances.

To my daughter and family

Acknowledgements

I should start by conveying my great appreciation to Mr. Luis Fariñas del Cerro, Director of Institute of Research on Informatics of Toulouse, who opened the doors of his institute and gave the chance of doing research there.

My sincere thanks to Professor Bruno Defude and Professor Mesut Razbonyali who were interested in reviewing my thesis and put their valuable efforts on enchancing it. Thanks to the members of the defence jury, Professor Serge Miranda, Professor Halis Puskulcu and Florence Sedes who were also very supportive in evaluating my works presented.

Special thanks to my advisor Professor Abdelkader Hameurlain who let me be creative but kept me on track. Even more importantly, he taught me to choose worthwhile problems and aim high. Other special thanks should be sent to my co-advisor, Professor Halil Sengonca who was always ready to support and without his help I could not write the Turkish version of my thesis. Thanks also to an exceptional chief, Mr. Franck Morvan with whom I've had countless stimulating discussions that led to new ideas.

I greatly appreciate the support that I got from my professor Sitki Aytac during these years which coincided with a difficult transition period in my personal life as well.

Thanks to the present and past members of team Pyramide for being supportive and friendly with me during all my stays. Special thanks to Nadhem Marsit without whom I would never feel at home and get the chance of contacting other people of IRIT.

Thanks to my friends who live in Turkey and abroad for their patience in listening to my changing moods and conveying their thrust with every chance. Special thanks to my students for keeping me motivated and young although I was unreachable periodically during those three years. I appreciate the support of Herkul and my pirate who were always reachable when I needed.

I am grateful to my parents who were always source of inspiration for me. They were with me whenever I needed them during these years. Thanks to the members of my family from different ages, they always made me feel cared and loved. Last but not the least, a thank you goes to my daughter for her support and confidence although during past three years I must have neglected her lot.

Table of Contents

List of Figures

Chapter I Introduction

1. Context

Many of the current applications like e-commerce, digital government, medical genetics, astrophysics, environment and culture require access to the data from different sources. The main reason behind this is the increase in deployment and use of web due to the technical advances achieved. This increase demand highlighted the role of database technology as well: in storing, managing and efficiently querying large amounts of scattered information besides the functions for protecting data integrity, facilitating reliable and flexible data access and manipulation, synchronizing concurrent accesses from applications and securing data.

Increase in the demand of sharing data from different sources resulted in the research efforts to provide integration of data scattered over different sources. The aim of **data integration** is to provide uniform access (same query interface to all sources) to multiple, distributed, autonomous and heterogeneous data sources. As stated by [OZS 99] **distribution** of data means that the data resides on different sites. The **autonomy** of a system indicates the distribution of control: the data source might be sharing part of its data but in terms of operation it acts independently. Data source may be autonomous in its design, meaning their developers are free to define the domain, programming models, naming concepts, etc. A data source may be autonomous with respect to communication and execution, meaning that it may independently decide how to handle interaction with the outside world and how to handle the demands internally [HAS 00]. Due to the independent development and deployment of data sources, **heterogeneity** might occur at various levels and for various reasons. On a technical level, heterogeneity comes from different hardware platforms, operating systems, database

management systems. On a conceptual level, heterogeneity comes from different data models as well as different understanding and modeling of the same real-world concepts, for example, the use of the same name to denote different concepts and the use of different names for the same concept.

Major dimensions to consider in data integration are the number of sources, amount of data volume to be transferred, heterogeneity and autonomy of the sources together with the network bound imposed by the distribution and the requirements of the users like accuracy, completeness, performance, handling inconsistencies.

In this chapter, we will first discuss on data integration approaches where we will mainly compare data warehousing approach with virtual data integration. In section 3 we will present the mediator wrapper architecture with the functions of its components, phases of query processing in this architecture and the issues related with limited query capabilities of the sources. In section 4 we will discuss and present the position of the problem that motivated us for the works presented in this thesis. Content and contributions of the thesis can be found in section 5 of this chapter.

2. Data Integration Approaches

Main current solutions for data integration are data warehousing and virtual data integration. We want to explain and compare the approaches of data warehousing and virtual data integration.

2.1 Data Warehousing

One popular approach is Data Warehousing as solution to enable a common storage providing user a uniform interface over different sources [GUP 97 and WID 95]. As seen in the Figure 1.1, data from several sources are extracted, transformed and loaded (ETL) into a source and can be queried with a single schema. This can be perceived architecturally as a

tightly coupled approach because the data reside together in a single repository located in one site. All user queries are posed to that common storage. The advantage of this system it is effective in querying and the decisional data is separated from operational data. However, problems with tight coupling can arise with the "freshness" of data, for example when an original data source is updated, but the warehouse still contains the older data and the ETL process needs to be executed again. It is not possible to add a new source. Another difficulty is to construct data warehouses when you only have a query interface to the data sources and no access to the full data.

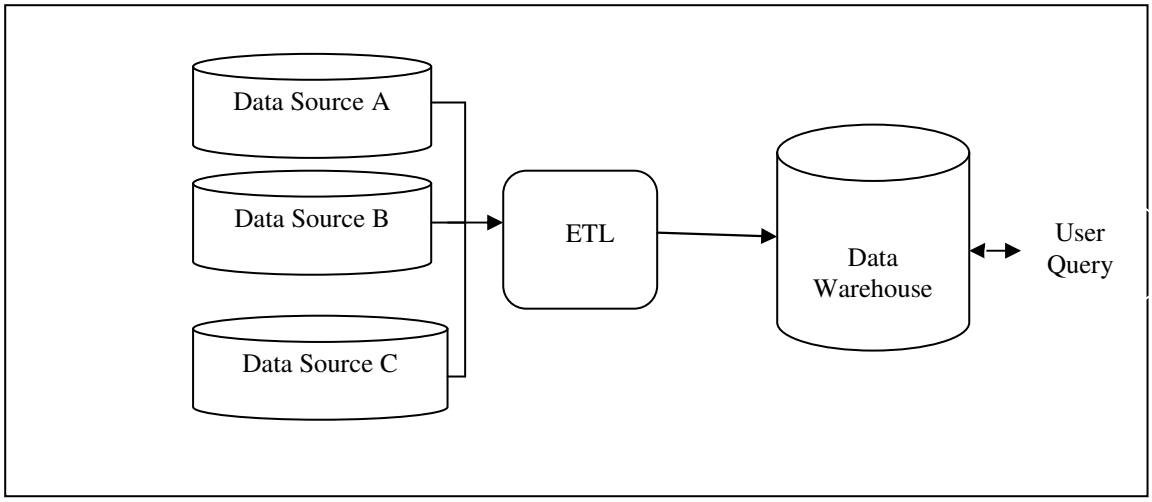

Figure 1.1: Simple schema for a data warehouse

2.2 Virtual Data Integration

The recent trend in data integration has been to loosen the coupling between data following the idea presented in [WIE 92]. Here the idea is to leave the data in the sources, when the query arrives; determine the relevant sources to the query, break down the query into sub-queries for the sources using the mediated schema, get the answers from the relevant wrappers and combine them appropriately in the mediator (Figure 1.2). This process can also be called as view based query answering because we can consider each of the data sources to be a view over the mediated schema.

There is a large body of research projects based on this approach: Garlic (IBM) [ROT 96], Information Manifold (AT&T) [LEV 96], Tsimmis [RAJ 95], InfoMaster (Stanford) [GEN 97], Tukwilla (UW) [IVE 99], Hermes (Maryland) [ADA 96], Disco (INRIA, France) [TOM 98], Cape [ZHU 04].

Other major component of this architecture is the wrapper whose task is to communicate with the data source and do format translations. Queries submitted in the language of mediator are translated into the language of the source and the results of the source are transformed into the format of the mediator. They are built with respect to a specific source and often it is hard to build. They can be intelligent in terms of performing source-specific optimization.

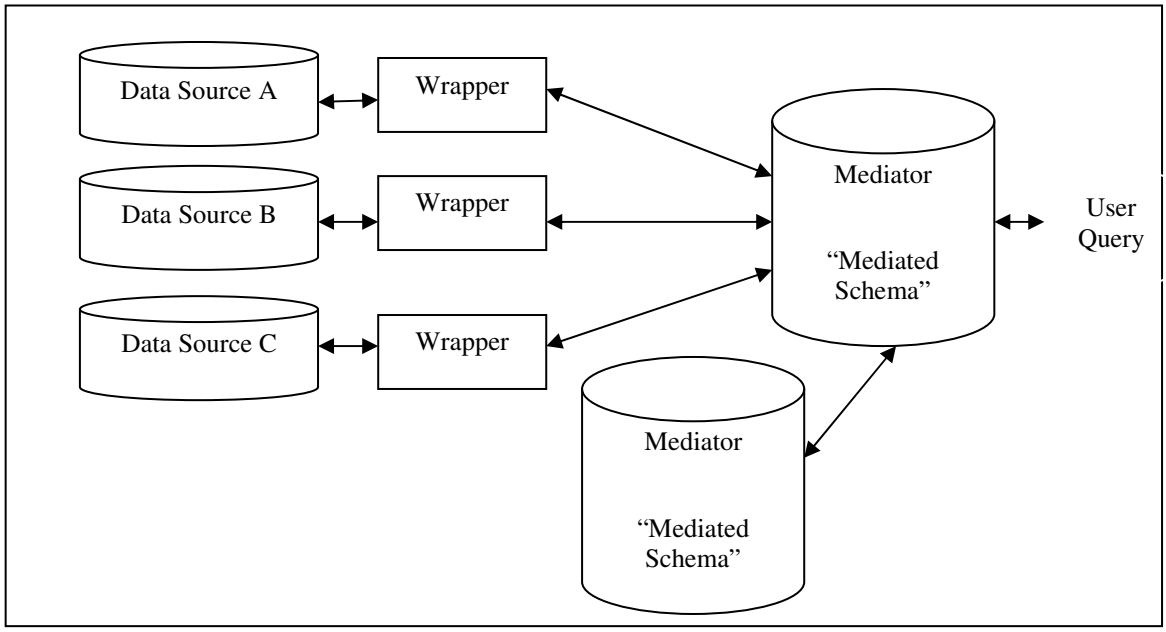

Figure 1.2: Simple schema for data integration solution

A convenience of this solution when compared with data warehouse solution is that new sources can be added by simply constructing a wrapper for them. This contrasts with ETL systems or a single database solution where the entire new dataset must be manually integrated into the system. Another advantage is that the data is always fresh. However, the challenges are expressive power, easy addition of a new source, effective and efficient

reformulation of the query for local data sources, query optimization and limited query capabilities of the sources. We will discuss these issues in section 3 of this chapter.

Throughout the rest of the thesis we will be referring to this approach when we say data integration, mediation system or mediator-wrapper architecture.

3. Query Processing in Data Integration Systems

Let us look at the steps and the components of the query processing in data integration. As seen in Figure 1.3, the main components of a data integration system are mediator and wrapper basically and we have already summarized the functions of them in previous sub-section. In mediator a user query goes through three successive operations; (1) query reformulation: user query given in global schema is transformed into the sub-queries written in local source schemas, (2) global optimization: execution plans of the sub-queries with optimum cost is found, (3) execution plan is submitted to the execution. We will discuss about describing source descriptions in sub-section 3.1 and continue with the explanation of the query reformulation in sub-section 3.2. In sub-section 3.3 we will present query optimization in data integration.

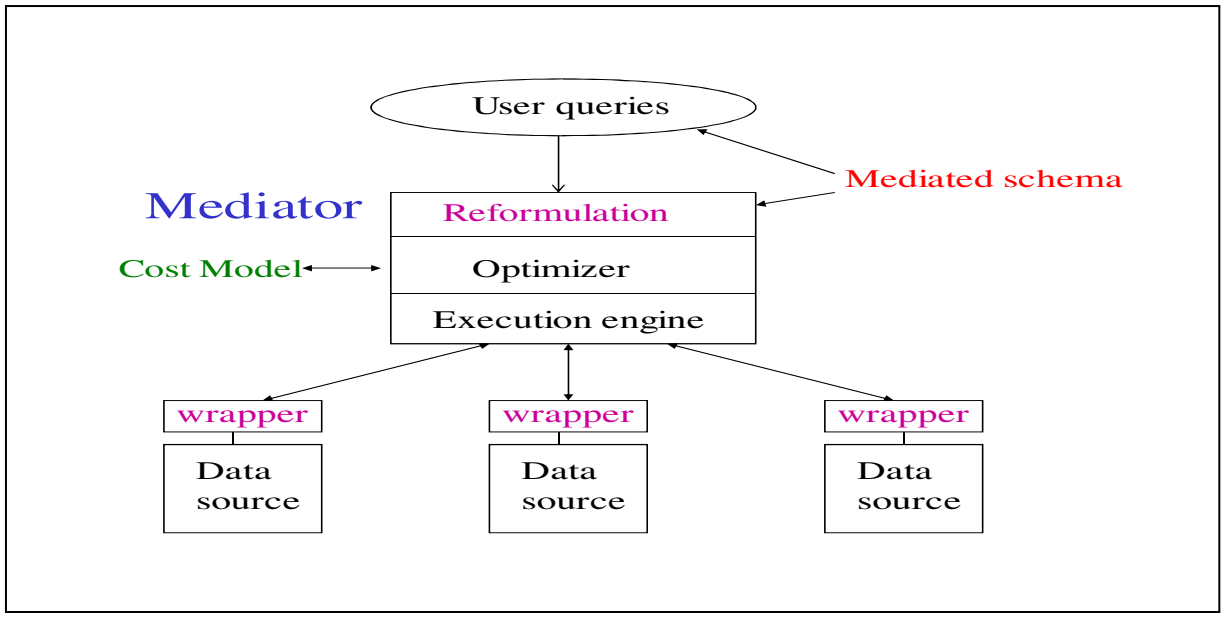

Figure 1.3: Query processing in data integration systems

3.1 Approaches to Specify Source Descriptions

[LEN 02] states that data integration systems are formally defined as a triple $\langle G, S, M \rangle$ where G is the global (or mediated) schema, S is the heterogeneous set of source schemas, and M is the mapping that maps queries between the source and the global schema. Both G and S are expressed in languages over alphabets comprised of symbols for each of their respective relations. The mapping M consists of assertions between queries over G and queries over S. When users pose queries over the data integration system, they pose queries over G and the mapping then asserts connections between the elements in the global schema and the source schemas.

A database over a schema is defined to be a set of sets, one for each relation (in a relational database). The database corresponding to the source schema S would be the set of tuples for each of the heterogeneous data sources and is called the source database. Note that this single source database may actually be a collection of disconnected databases. The database corresponding to the virtual **mediated schema** G is called the global database. The global database must satisfy the mapping M with respect to the source database. The legality of this mapping depends on the nature of the correspondence between G and S. Two popular ways to model this correspondence are Global as View or GAV and Local as View or LAV.

In GAV, the global database is modeled as a set of views over S. In this case M associates to each element of G a query over S. Query processing becomes a straightforward operation because the associations between G and S are well-defined. The burden of complexity is placed on implementing mediator code instructing the data integration system exactly how to retrieve elements from the source databases. If any new sources are added to the system, considerable effort may be necessary to update the mediator, thus the GAV approach should be favored in cases where the sources are not likely to change.

On the other hand, in LAV, the source database is modeled as a set of views over G. In this case M associates to each element of S a query over G. Here the exact associations between G and S are no longer well-defined. As is illustrated in the next section, the burden of determining how to retrieve elements from the sources is placed on the query processor. The benefit of an LAV modeling is that new sources can be added with far less work than in a

GAV system, thus the LAV approach should be favored in cases where the mediated schema is not likely to change.

For conclusion remarks we can state that in GAV approach query reformulation becomes as simple as query unfolding on the other hand in LAV it is a problem of answering queries using views and there exists a large body of research like Bucket algorithm of [LEV 96], inverse rules algorithm of [DUS 98], hybrid versions MiniCon [POT 01], etc.

3.2 Query Reformulation

The theory of query processing in data integration systems is commonly expressed using conjunctive queries [ULL 97]. One can loosely think of a conjunctive query as a logical function applied to the relations of a database such as "f(A,B) where A < B". If a tuple or set of tuples is substituted into the rule and satisfies it (makes it true), then we consider that tuple as part of the set of answers in the query. While formal languages like Datalog [KOW 88] express these queries concisely and without ambiguity, common SQL queries are classified as conjunctive queries as well.

An important property of conjunctive queries (in terms of data integration) is *query containment*. A query *A* contains another query *B* (denoted $A \supset B$) if the results of applying *B* are a subset of the results of applying *A* for any database. The two queries are said to be equivalent if the resulting sets are equal for any database. This is important because in both GAV and LAV systems, the user's conjunctive queries are posed over a mediated schema represented by a set of views, or "materialized" conjunctive queries. Integration seeks to rewrite the queries represented by the views to make their results equivalent or maximally contained by our user's query. This corresponds to the problem of answering queries using views (AQUV) [HAL 01].

In GAV systems, a system designer writes mediator code to define the query rewriting. Each element in the user's query corresponds to a substitution rule just as each element in the global schema corresponds to a query over the source. Query processing is simply expanding the sub-goals of the user's query according to the rule specified in the mediator and thus the resulting query is likely to be equivalent. While the majority of the

work is done beforehand by the designer, some GAV systems such as TSIMMIS involve simplifying the mediator description process.

In LAV systems, queries undergo a more radical process of rewriting. This is because there is no mediator to align the user's query with a simple expansion strategy. The integration system must execute a search over the space of possible queries in order to find the best rewrite. The resulting rewrite may not be an equivalent query but maximally contained, and the resulting tuples may be incomplete. For the problem of query reformulation, there exists a large body of research like Bucket algorithm of [LEV 96], inverse rules algorithm of [DUS 98], hybrid versions MiniCon [POT 01], etc.

3.3 Query Optimization

In traditional optimization, query optimization follows the steps of parsing, query rewriting (apply algebraic transformations, merge joins into blocks and move predicates before blocks), optimization of each block (join ordering), select scheduling. In this environment each relation is coming from a single database, all sources are assumed to be relational, completeness is expected and deterministic execution is possible since the statistics are available. Formal description of optimization is stated as "Given a query Q, an execution space E, and a cost function C defined over E, find an execution e in E_Q that is of minimum cost, where E_Q is the subset of E that computes Q, $\min_{e \in EQ} C(e)$" by [DU 92]. Any solution to the above problem can be characterized by choosing: (1) an execution model and, therefore, an execution space, (2) a cost model, and (3) a search strategy. The execution model encodes the decisions regarding the ordering of the joins, join methods, access methods, materialization strategy, etc. The cost model computes the execution cost. The search strategy is used to enumerate the execution space, while the minimum cost execution is being discovered. These three choices are not independent; the choice of one can affect the others.

The **execution space** used can be abstractly modeled as the set of all join orderings (i.e., all permutations of relations) in which each relation is annotated with access/join methods and other such inflections to the execution [OZS 99]. It is obtained by applying transformation rules (e.g. commutativity of joins, associativity of joins, permutation of two

operators, etc.). It is usually large in size; the size depends on the number of relations and form of the query (e.g. chain, star, etc.). In order to reduce the size of the execution space optimizer might apply some heuristics like deconstructing conjunctive selections into a sequence of single selection operations, moving selection operations down the query tree for the earliest possible execution, avoiding cartesian product operations, creating new projections where needed, identifying those sub-trees whose operations can be pipelined, etc. Three types of execution tree structures are demonstrated in Figure 1.4.

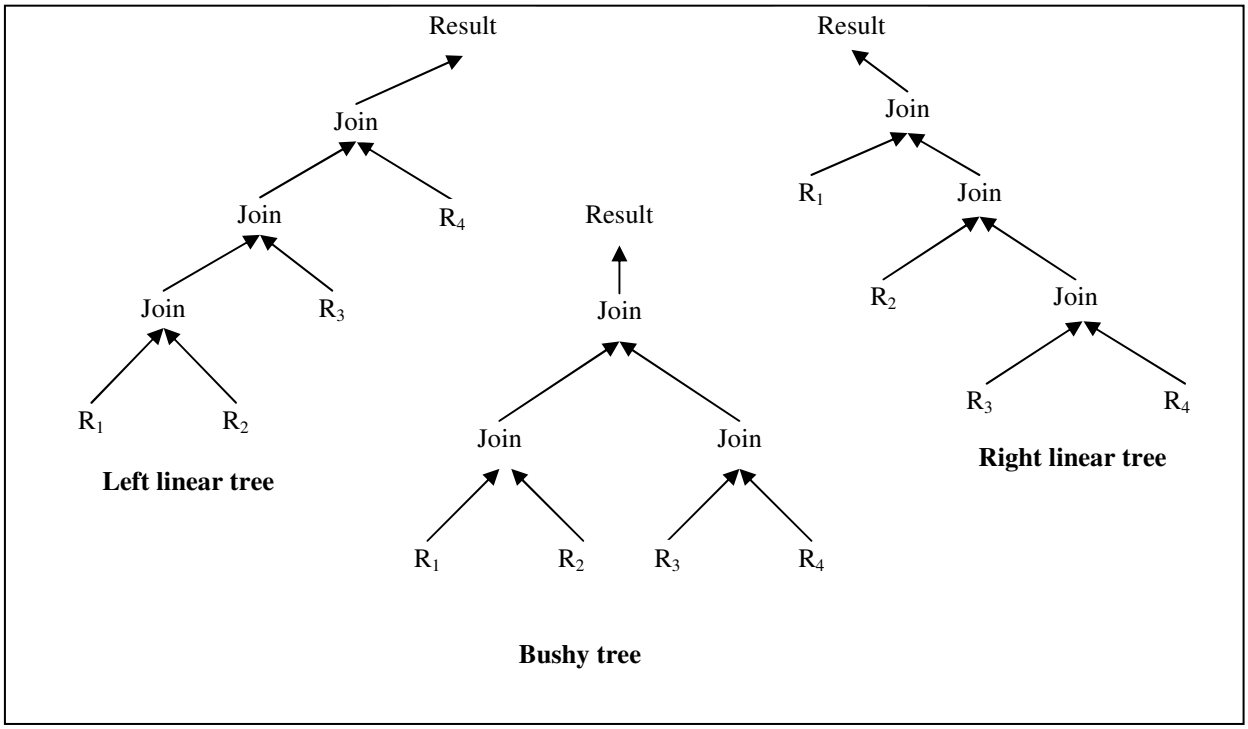

Figure 1.4: Structures of query execution trees

When it comes to **cost model** we can say that it is used by the search strategy of the optimizer to find the optimum plan. The cost model is composed of the statistics about the data and the cost formula. The cost of a plan is calculated by the costs of operations in the query together with the cost of data transfer between the sites. The cost of an operation is the cost of processing (CPU) and the cost of I/O. In order to estimate the cost of an operation, cost formula uses the estimates on the statistics. The weights of these costs can be changing according to environment. In data integration environment, it is difficult to estimate the costs

of the operations since the statistics about the data and the cost formula of different sources can be obsolete or unavailable. Different cost models are proposed by the researchers in order to overcome this problem. We will be discussing about these methods in chapter 2.

The **search strategy** of an optimizer is the way of the optimizer exploring in the execution space in order to find the optimum execution plan. There are different strategies behind the search algorithms: enumerative or randomized (e.g. genetic) [LAN 91]. Enumerative search strategy considers most of the points in the solution space, but tries to reduce the solution space by applying heuristics. The System R optimizer [SEL 79] demonstrates this approach by restricting the solution space to binary processing trees and using dynamic programming for searching. Enumerative strategies can lead to the best possible solution, but face a combinatorial explosion for complex queries (e.g. a join query with more than ten relations).

In order to consider larger execution spaces, randomized search strategies have been proposed based on the idea to improve a start solution until obtaining a local optimum. Examples of such strategies are simulated-annealing [IOA 87] and iterative improvement [SWA 88]. With the same objective, genetic search strategies [GOL 89] can be applied to query optimization as well. Randomized or genetic strategies do not guarantee that the best solution is obtained, but avoid the high cost of optimization.

3.4 Limited Query Capabilities of the Sources

Now we want to introduce the aspect of limited query capability; which is one of the characteristics of the most of the sources in data integration. Consider a web book store where user poses queries to this source by filling out a search form: author, title, subject, ISBN, publisher attributes. Specifically user must fill out at least one of the five attributes and submit the form. In response the source will return a list of books with some of these and additional attributes. In particular, it does not return the subject attribute but returns the price attribute among the others. Many simple queries cannot be posed to this source; e.g. books written by “Freud” that cost at most 10 Euro. This source has limited query capability in another words, is a **restricted source**. Sources might be restricted due to the limitations of its query interface,

certain attributes might be hidden due to privacy reasons or it mains indexes for some attributes only.

Let us give another example; consider a web bookstore, BarnesAndNoble.com [BAR 01]. Once again users can pose queries by filling out a search form which allows users to specify at least one of the author, title or keyword attributes along with optional specification of price, format and other attributes. In particular that web bookstore can support the query to find all by “Freud” that cost at most 10 Euros. However, it cannot support many other simple queries like “find all books published by Mc Graw Hill”. Incidentally, this query is supported by Amazon.

As seen by two examples query optimizer in data integration systems needs mechanisms to handle capability limitations together with capability descriptions (e.g. views). A language or a pattern should be found to describe these limited capabilities [MAN 01, VAS 97]. Query decomposition methods of the optimizer should be revised to take into account these restrictions like the case of [DUS 98, LI 03 and MAN 01] in finding executable plans (supported queries by the sources). [LI 03] differs from [MAN 01] since it incorporates the sources that are not specified in the query to get maximum or partial information from the restricted sources as well. Feasible plan within this execution space should be found like the case of positive query (query posed on the sources with no access restrictions) [DUS 98, MAN 01, YER 99]. In query execution, additional operators are required to exploit a source with limited access [BOU 01, CHA 93 and MAN 01].

We have gone through phases query optimization; when the environment is data integration. In the next section we want to position the problems that motivate us in the framework of this thesis.

4. Problem Position

Up to this point, we have introduced the need for accessing the data residing on different sources which are distributed, autonomous and heterogeneous. Among possible solutions for data integration that would provide user with uniform and transparent access to

these sources; we have introduced mediator-wrapper architecture. As we focused on the query processing in this architecture we have outlined various issues and research problems like describing source capabilities, query reformulation, query optimization, limited query capabilities of the sources. In this thesis, we want to concentrate on query optimization in data integration environment with the presence of the sources with limited query capabilities.

The focus of the first group of research on query optimization in data integration is related with finding better cost estimations while handling the autonomy of the data sources from which it is difficult to provide statistical or cost information. In these efforts; the **cost model** used in optimization is extended as to incorporate the estimations based on the result of the similar historical queries, result of calibrating or probing queries, cost models of the data sources [ADA 96, AMB 98, GAR 96, NAA 98 and ZHU 03]. The idea behind all these efforts is to better estimate the cost of an execution plan in compile time where the statistics might be absent or erroneous and knowledge about execution environment is limited.

However these cost models face the challenge of producing sub-optimal execution plans due to the unpredictability of the sources and execution environment. General solution to this problem is **adaptive query optimization** meaning interleaving optimization and execution (as you get to know more about your data and environment, you can improve your plan). Query optimizer has to generate stable plans to deal with execution uncertainties; non-blocking plans which give out results as soon as they are available. [AMS 97, AVN 00, KAB 98 and ZHO 05] are some of the examples of dynamic optimization methods; where the idea is to produce an execution plan in compile time, execute and when errors on the estimations occur, react as to adapt to run- time conditions. In all these examples, there is a main process in charge of monitoring and changing the execution plan during query execution. This kind of monitoring and decision taking is classified as **centralized** re-optimization by [HUS 06]. The drawback of these methods comes when they are applied to large scale environment due to the characteristics of the large scale environment where there is high network latency and low bandwidth. High amount of control and data message passing (1) creates bottleneck on the site of central authority and (2) increases the cost of the query due to high transfer cost.

In order to overcome this bottleneck, there are recent studies where the monitoring and decision taking for re-optimization is **decentralized.** The examples of such methods are presented in [ARC 04, COL 04, JON 03, IVE 99, MOR 03, OZA 05a and URH 00]. The approach of [IVE 99 and URH 00] is decentralized: the decision of adaptivity is taken by the operator. [JON 03] and [COL 04] are the examples of decentralized optimization methods where the adaptivity is at the sub-query level and execution model is based on communicating agents or brokers. These methods concentrate on improving the cost of local processing by changing the plan at operator level or at sub-query level in the presence of inaccuracies but do not take into account the network bound.

The methods presented in [ARC 04, HUS 05, MOR 03 and OZA 05a] are based on the idea of using mobile relational operators in query execution. In this **mobile query execution model**; relational operators are mobile agents which are aware of their environment and able to react in an autonomous and decentralized way during their execution to correct the sub-optimality of the execution plans. The mobile relational operator can migrate from one site to another among the set of sites determined by the optimizer, according to workload of the sites, the data unavailability, statistical information and the cost of migration. This style of execution allows taking into account the cost of local processing together with network sources. However, all the previous study on mobile query execution model is based on the mobile relational operators for positive query (query posed to sources without access restrictions).

As in the examples given in section 1.3.4 in large scale data integration environment some of the data sources have limited query capabilities. In order to retrieve data from them some of the required input should be given When dealing with these restricted sources data integration systems need (1) to model descriptions of the query capabilities of each source, i.e., the set of queries supported by each source [KIR 95 and MAN 01], (2) to have algorithms for deciding how a query can be answered given the capabilities of the sources at a minimum cost [LI 03, MAN 01, NIE 01, MOR 88 and YER 99] and (3) to device special algorithms for the operators in the query execution process [GOL 00 and MAN 01]. [MAN 01] is an elegant and compact work covering all the issues related with restricted sources: modeling the limited source capabilities, analyzing effects of the restrictions on the size of the

execution space, finding viable and feasible plan algorithms and special operators for execution. Bind Join operator of [MAN 01] focuses on returning the answer as early as possible but it is not adaptive.

Mobile query execution model as an example of decentralized adaptive query execution should include operators in order to query sources with restricted sources. Another drawback of mobile query execution model lies in it assumption that the optimizer can map the mobile relational to a site which will provide the optimum execution with single point estimations. However, mobile relational operators should be placed to the sites which would not obstruct them to move at run-time with refreshed estimations when there is an estimation error. Studies capturing the problem of building bridge between compile-time optimization methods and run-time adaptive methods are critical at this point. The approach presented as proactive re-optimization in [BAB 05b] is an elegant example of such studies. The authors are trying to find robust execution plans within bounding boxes around estimates that define the uncertainty in estimates of statistics. Similar to this approach, initial placement of mobile relational operators should take account the possibility of their migration in case of estimation errors. When we look at the placement methods provided by the parallel and distributed query processing field [CHE 92, CHE 95, FRA 98, GAR 96b, HAS 94, HON 91, LIU 98, LO 93, MAC 86 and ZHO 05] are based on single-point estimations. Such a placement is as good as compile-time estimations but it yields bad performance in cases of estimation errors. When we use a mobile relational operator, its placement is critical since wrong placement can block its adaptation capability.

5. Content, Contribution and the Organization of the Thesis

In this thesis we start by presenting the **context** and try describing the challenges in terms of data integration. The challenges arise from the characteristics of the data sources, characteristics of the execution environment and recent user requirements. The sources to be integrated are heterogeneous, autonomous and distributed. In terms of query optimization all three attributes pose variety of problems. Heterogeneity necessitates methods to describe sources capabilities and query limitations, methods to decompose the query into sub-queries of the sources. Autonomy requires cost models to capture the different data sources in the

global cost of the query and adaptive re-optimization methods in case of estimation errors. Distribution and execution environment need extra attention to be paid to the volume of data and control message passing and to the occurrence of unpredictable events during execution of the query. Recent user requirements changes the objective of query optimization and forces the addition of new metrics like early arrival of the tuples, monetary costs of the data sources, quality of data sources, etc.

As seen in the context there is large spectrum of challenges; our interest was to focus on the query optimization in large scale data integration environment. Within this objective, our initial effort was to make a survey on the **state of the art** concerning the issues of query optimization in large scale data integration and locate our motivation. In this survey we first focused on cost models where the main concern was to be able to make better cost estimation about the global cost of the query capturing necessary information from different data sources. We have seen that, it was difficult to make good estimation at compile time due to the autonomy of the data sources and occurrence of the unpredictable events during execution time. To overcome these drawbacks, large amount of research were carried out in the area of adaptive query optimization. We at this point, made analysis of the proposed adaptive query optimization methods based on attributes like type of modification, place of modification, type of control, type of event and type of re-optimization. We found one of the attributes critical in large scale: the control of monitoring and decision making about adaptation. We emphasized the importance of decentralized control and demonstrated various works in this direction. We focused on mobile query execution model and its building blocks studied so far and seen that it needs methods to handle query execution (1) with the sources that have limited query capabilities and (2) to have link with the optimizer in mapping the mobile relational operators to the sites regarding their sensitivity to initial placement. This brought us to the survey of the methods of query processing in the presence of restricted sources. Finally we were ready to propose our methods for mobile query execution model in the presence of restricted sources and initial placement of mobile relational operators.

In our first work, our challenge was to propose **mobile join operators designed for restricted sources**; meaning they should be able to work in the presence of access restrictions imposed by participating data sources. This effort would extend the power of mobile query

execution model which is worth considering with its decentralized adaptive query optimization approach applicable in large scale network. In this perspective, we designed Mobile Dependent Join (MDJoin) and Sampling Dependent Join (SMDJoin): two join operators which are mobile (they adapt to inaccuracies at run time by changing their place of execution) and able to work with access restrictions imposed. The difference between the two new query operators lies in their level of adaptation ability to the execution environment. In our second work, we addressed the problem of **initial placement** taking into account the adaptation capability of mobile relational operators at run-time (mobile join, MDJoin and SMDJoin). Our challenge was not finding good placement based on single-point estimates but defining acceptable placement for an estimation interval in order to avoid dramatic performance at run-time.

We have carried out a **performance evaluation study** on our methods: mobile relational operators of restricted sources and initial placement of mobile relational operators. We had done our performance evaluation of both of the methods in a simulation environment. First study evaluates the performances MDJoin and SMDJoin in comparison to the dependent join operator of the standard strategy. The focus of the evaluation was on their performance with respect to the following parameters: response time in the presence of the variations between the compile-time estimations and the run-time computations of the sizes of the relations, speedup[1] realized in comparison with the network bandwidth and CPU frequency. The second study compares single-point and robust placement methods mapping single join and multi-join. Both methods were tested when there is error on the estimations related to the sizes of relations.

The **contribution** of the thesis is as follows; (1) two mobile join operators (MDJoin and SMDJoin) which are capable to work with the sources having limited query capabilities and yet capable to adapt to run time conditions when there is estimation error and (2) initial placement method which finds robust site to place mobile relational operators taking into account their migration (adaptation) possibility. Performance study on (1) showed that if the

[1] The speedup: response time of a DJoin / response time of a mobile join operator

size of the data from the first source is more than 30% different than the estimation made by the optimizer, our mobile dependent operators have better performance compared to the operator of the standard strategy. Against the variations of the size of the data coming from the second source SMDJoin is superior to MDJoin, since due to its sampling step it sees the variation of the size of the data of the second source. The overhead of this sampling step for SMDJoin is less than 10 % of increase in response time compared to MDJoin. The performance study on (2) showed that robust placement method has better performance in case of the error on the size of the first relation 80%. We noticed that the error on the size estimated at compile-time of the first relation has greater impact on the query response time. Robust placement method outperforms when there is error on the size of the first relation since it expects adaptation and chooses the initial site according to this possibility.

Plan of the rest of the thesis is as follows; in chapter 2 we will give a state of art covering our concerns related with query optimization in large scale network. In chapter 3 we will present two contributing studies of the thesis: mobile relational operators for restricted sources and initial placement of mobile relational operators. Chapter 4 is dedicated to performance results of the works discussed in chapter 3. Chapter 4 will present our conclusion remarks and the ideas for future work.

Chapter II State of the Art: Query Optimization in Data Integration Systems

1. Introduction

In previous chapter we explained the need and the idea behind data integration and characteristics of the sources. As to brief; we can state that data integration approaches try to provide user uniform and transparent access to the data sources which are heterogeneous, autonomous and distributed over large scale network. Large spectrum of challenges arises due to the characteristics of the sources, environment and user requirements. First four of the challenges explained below are related with the characteristics of the data sources in data integration systems whereas 5 and 6 is related with the environment of execution and the last one with the user requirements.

1) Heterogeneity: As explained earlier, heterogeneity can happen at different levels of the data integration system. Bridging that heterogeneity in terms of source descriptions, query reformulation, cost models, etc. may have important impacts on the optimization process.

2) Autonomy: Autonomy, has a more serious impact since several optimization methods require specific information e.g. statistics, cost formula from information sources. This information is not always easily available.

3) Distribution: Distribution is another challenge on query optimization; data is scattered over large number of sources. The large scale network imposes restrictions in terms of network latency and bandwidth. Query optimization should be aware of the possibility of the cases with high amount of data and control message passing required since this may end up in bottleneck towards a central site or congestion of network due to the network bound.

4) Restrictions: Data sources might not have the same query capabilities. The restrictions might be due to security and performance reasons. Query optimization in data

integration systems require means to model, to find efficient and viable plans and to have operators capable to work with these sources.

5) Dynamic environment: Unpredictable events (e.g. change in the availability of the memory, CPU load) could happen anytime during query execution. The query optimizer would need mechanisms to avoid missing optimal query execution in the occurrence of an event; e.g. means to gather information and methods to react (modify the plan) in execution time.

6) Number of sources: query optimization should take into account scalability issues to avoid performance degradation. This degradation could lead to very inefficient query execution plans.

7) Optimization Paradigm: This is related with user requirements in data integration; in traditional optimization as explained in previous chapter, main concern of optimization is to minimize response time. However optimizer of data integration systems might have to deal with parameters like fees of the sources, quality of data (e.g. freshness), etc. besides response time in cost function.

First and last challenge is out of the scope of this thesis. In this chapter we will make a survey on the state of the art related with remaining challenges. In the first part, section 2, we will focus on the research related with the cost models used in data integration systems where the objective is to capture information from data sources in order to make a better estimation at compile-time for the cost of the queries. In section 3 we started by presenting our classification parameters of adaptive optimization methods and continued discussing adaptive query optimization methods with inter and intra operator adaptivity. We preferred to dedicate a separate section (section 4) to explain the query optimization in the presence of restricted sources since the characteristics of these sources force us to concentrate on modified methods in modeling the restrictions, execution space generation, optimization algorithms and execution operators. In section 5 we have discussion on the applicability of the adaptive optimization methods in large scale data integration. Section 6 is dedicated to the explanation of previous studies on mobile query execution model. In section 7 we wrap up the chapter with our final remarks and introduction of the ideas motivating the works presented in following chapters.

2. Cost Models in Data Integration

Several researches in data integration systems focus on the cost models extending the classical cost model in order to capture data sources. The main idea behind these efforts is to better estimate the cost of the global query incorporating the information retrieved from the autonomous data sources. They extend the classical cost model by various means such as estimation based on past similar queries, estimation based on probing queries or estimation incorporating cost estimations of the wrappers, in cost estimation of the global query. The methods can be classified according to the cost model they use as historical [ADA 96], calibrating [DU 92, GAR 96, ZHU 98, ZHU 00 and ZHU 03] or generic [NAA 98, NAA 99, ROT 99].

2.1 Historical Cost Model

Main idea in historical cost model presented in Hermes [ADA 96] is to accept that the formula based on execution costs is an accurate estimate for the next identical sub-query. HERMES stores the statistics of the calls to the data sources by recording the selection predicate. These statistics are exploited during the optimization of the next similar query (i.e. for identical selection predicate) on the same sources. This cost model is efficient for similar queries and when the data volume does not vary too much between two calls. In case the selection predicate change or during a first call to a data source, the cost model is “blind”. Furthermore, in the transactional applications where the data volume associated to a selection predicate can be different between two invocations, this model becomes less efficient.

2.2 Calibration Cost Model

The idea behind calibration cost model is to use a generic cost model in query optimization in the mediator [DU 92] and get the estimates from the wrappers for the coefficient of the parameters in the cost formula of this model. The process of calibration is realized once for each class of systems. [GAR 96] extended this approach as to capture object-oriented database systems. This approach has been implemented in Pegasus [SHA 93] and in IRO-DB [GAR 95]. [ZHU 98] proposed a query-sampling method that develops regression cost models for local query classes based on observed costs of sample queries. The idea is to

group the similar type of queries into classes and for each class, to make the calibration with a sample query in this class. In this approach, in order to group the queries into classes the characteristics of the referenced relations have to be known. This approach is extended to take into account the dynamic parameters related with the environment e.g., CPU load, available memory [ZHU 00, ZHU 03]. The drawback of the calibrating approach appears when the data source does not follow the generic cost model imposed by the mediator.

2.3 Generic Cost Model

Generic cost model introduced by [NAA 98, NAA 99] combines a generic cost model with specific information exported by the wrappers. The wrapper can specify any part of the cost information. This cost model is implemented in DISCO [TOM 98]. [ROT 99] is another example of this approach used in GARLIC data integration system. Mediator uses a default cost model and wrappers cooperate in the estimation of the total cost of a given query plan. Wrappers may use a default cost model or a more specific one to model their execution strategies. The default cost model considers the total cost of a POP operator as a combination of the cost of two basic tasks: reset for initializing the operator and advance for retrieving the next result. Default cost formulas are used to compute these two costs based on statistics. Based on the cost of the reset and advance tasks, the wrapper should be able to provide cost estimates for its own plans that are included in the global query plan. The total cost, re-execution cost, and result cardinality are considered in the cost of a PushDown POP. They are required to estimate the cost of the global query plan. Once the optimizer selects a winning plan, that plan is translated into an executable form. However, this approach is highly dependent on the developer of the wrapper.

3. Adaptive Query Optimization

All the research mentioned in previous section focus on the cost model in data integration in order to better estimate the cost of the global query. Their effort is to retrieve statistical or cost model information from data sources while estimating the cost of the query at compile-time in order to find the optimum plan. But this optimum plan faces the challenge of becoming sub-optimal during run-time basically due to (1) missing or unavailable

statistical information about the data or the cost model of the autonomous sources at compile time and (2) the occurrence of unpredictable events during execution time. The challenge to overcome these problems caused the development of numerous adaptive (dynamic) query optimization methods. They generally start with an optimized execution plan, try to change execution plan at run-time with better knowledge of data and run time conditions (e.g. re-optimize the query or use specific operators to deal more flexibly with unpredictable events). They address the efficiency in the presence of unavailable or undependable statistics, unavailable cost models of data sources and unpredictable events (e.g. delays in data arrival, change in CPU load, etc) during query execution. Characteristics of such methods are given in [HEL 00]. It states that a query optimization is adaptive if: (1) it receives information from its environment, (2) it uses this information to determine its behavior, and (3) this process iterates over time, generating feedback loop between environment and behavior.

3.1 Classification of Adaptive Query Optimization Methods

Based on the definition of adaptivity, we want to analyze the methods depending on the following attributes: the level of adaptation, type of event triggering the modification, type of modification realized, type of control about decision making, type of re-optimization and the environment (Table 2.1).

Level of modification (1) indicates the place of the modification of the method: inter-operator and intra-operator. **Type of event (2)** shows the case triggering the decision: presence of estimation errors on statistics, memory unavailability (memory allocated for the operators may become unavailable during execution), delays in data arrival (delay in arrival of data from remote sources due to network congestion or bandwidth of network) and user preferences (e.g. first tuple first). **Type of the modification (3)** can be re-optimization (re-optimize the rest of the execution plan with respect to run-time conditions) or re-scheduling (change the schedule of the operators for the rest of the execution) or replacement (change the implementation of the operators, e.g. replace hash join by sort merge join) or dynamic operator (operator is capable to adapt).

Method	Level of Modification 1	Type of Event 2	Type of Modification 3	Type of Control [HUS 06] 4	Type of Re-optimization [BAB 05a] 5	Env. 6
Mid-Query Re-Optimization [KAB 98]	inter-operator	estimation errors	re-optimization	centralized	reactive	M
Query Scrambling [AMS 97]	inter-operator	delays	re-optimization	centralized	reactive	M
Tukwilla / ECA rules [IVE 99]	inter-operator/	all	re-optimization	centralized	reactive	D
Progressive Query Optimization [MAR 04, KAC 06, HAN 07]	inter-operator	estimation errors	re-optimization	centralized	reactive	M, D, P
Proactive Re-Optimization [BAB 05b]	inter-operator	estimation errors	re-optimization	centralized	proactive	M
TELEGRAPH "Eddy" [AVN 00, BIZ 05, ZHO 05]	intra-operator	all	re-scheduling	centralized	reactive	M/ D
Tukwilla/Adaptive Data Partioning [IVE 04]	intra-operator	estimation errors	re-optimization	centralized	reactive	D
Tukwilla / Double Hash Join [IVE 99]	intra-operator	all	dynamic operator	decentralized	reactive	D
Xjoin [URH 00]	intra-operator	delays/user preferences	dynamic operator	decentralized	reactive	M
Broker-Based [COL 04]	inter-operator	all	re-optimization	decentralized	reactive	D
Agent-Based [JON 03]	inter-operator	estimation errors	re-optimization	decentralized	reactive	D
Mobile Query Execution Model [ARC 04, MOR 03]	inter-operator/ intra-operator	all	dynamic operator	decentralized	reactive	D

Table 2.1: Comparison of adaptive query optimization methods

Type of the authority making the decision of modification: if a central **control (4)** for monitoring the run-time conditions and making the decision of adaptivity exists then the method is called centralized, otherwise it is decentralized method [HUS 06]. **Type of re-optimization (5)** can be reactive or proactive [BAB 05a]. By re-active re-optimization it is stated that the query processing follows the cycle of optimize-execute-make decision about adaptation (re-optimize). By proactive re-optimization it is understood that in optimization

phase the potential need for adaptation is considered and optimization captures the concerns of reducing the cost and frequency of adaptation. Last column indicates the **environment (6)** for which the method is applicable, three types are possible: M (mono-processor), D (distributed) and P (parallel).

3.2 Adaptive Query Optimization Methods

There is a large body of research on adaptive query processing in data integration systems and some surveys trying to compare these works with respect to different characteristics. In this section we will look over some methods of adaptive query optimization. Our analysis is done with attention of the place of adaptation: inter or intra operator. All the methods discussed in the following sub-section are centralized methods: the monitoring and decision taking is done by a central control. In section 5, we will demonstrate some decentralized adaptation methods while discussing the applicability of centralized methods on large scale network.

3.2.1 Inter-operator

The works [AMS 97, KAB 98, IVE 99 and MAR 04] we will present in this section, propose to modify execution plans after the termination of the execution of an operator or after the materialization of a temporary relation. The type of modification is re-optimization, re-scheduling or replacement for the remainder of the execution plan.

The goal of **query scrambling** of [AMS 97] is to react to delays by modifying the query execution plan on-the-fly. Unexpected delays (initial delay, bursty arrival or slow delivery) are “hidden” by performing other useful works. Scrambling has a two-pronged action: rescheduling (scheduling other operators for execution) and operator synthesis (new operators are created when there is no other operator to execute). These two techniques are repeated as necessary to modify the query execution plan. Scrambling policies differ by the degree of parallelism they introduce or the aggressiveness with which scrambling changes the existing query plan. Three important trade-offs must be considered in re-scheduling: (1) the number of operators to reschedule concurrently (the benefits of overlapping multiple delays

and the cost of materializations used to achieve this overlapping), (2) rescheduling individual operators or entire sub-trees and (3) choice of specific operator(s) to reschedule. For operator synthesis, a significant amount of additional work may be added since the operations were not originally chosen by the optimizer. To avoid this problem a simple heuristic of avoiding Cartesian products to prevent the creation of overly expensive joins is used. However, the performance of this heuristic is highly sensitive to the cardinality of the new operators created.

In **mid-query re-optimization** method presented by [KAB 98], the execution plans are generated at compile time are annotated by the statistics used by the optimizer. At specific points, statistics collector operators are added to the execution plan. These operators calculate the statistics about the size and data distributions of temporary relations at run-time. The statistics are compared with those of marked by the optimizer in order to detect the sub-optimality. Correction of the sub-optimality is done in two ways: by dynamic re-allocation of the memory and by changing the remainder of the execution plan. They also demonstrate a technique to keep the overhead of this type execution within allowable limits.

Tukwila is another system addressing adaptiveness in data integration environment [IVE 99]. Adaptiveness is introduced at two levels: (1) between the optimizer and the execution engine, and (2) within the execution engine. In the first level, adaptiveness is deployed by annotating initial query plans by (event-condition-action) ***ECA rules*****.** These rules check some conditions when certain events occur and subsequently trigger the execution of some actions. Examples of events include operator failure, time-out, and out of memory exceptions. Conditions include the comparison of actual cardinalities known at run-time and those estimated. Finally, actions include rescheduling of the query operator tree, re-optimization of the plan, and alteration of memory allocation. A plan is organized into a partially ordered set of fragments and a set of corresponding rules. Fragments are pipelined units of operators. When a fragment terminates, results are materialized and the rest of the plan can be re-optimized or rescheduled. In addition to data manipulation, operators perform two actions: statistics gathering for the optimizer, and event handler invocation in case a significant event occurs.

In [MAR 04, KAC 06 and HAN 07] authors propose an optimization method called **progressive query optimization** (POP). During initial compilation POP (1) determines estimated parameters (e.g. cardinality) to hold if the plan is to be optimal one, (2) defines a validity range around it and (3) places CHECK operators at strategic points for validation of the actual cardinality during run-time. During run-time if estimated parameter is found to be out of the defined interval; all intermediate results are fully materialized, execution points are retained and optimizer is called again. At this point optimizer decides to use or not the already retained results on cost basis. Even if the partial results are not used related statistics are kept. This round of re-optimization is repeated whenever necessary at run-time. [MAR 04] is for local queries, [KAC 06] extends this idea for federated queries and presents POP/FED. And a recent work comes by [HAN 07] where POP is implemented in parallel environment with parallel checkpoints and parallel check point processing methods using communication protocols.

The four above methods presented are qualified as reactive re-optimization by [BAB 05a]. They use a traditional optimizer to generate an optimal execution plan; they collect statistics at run-time and make modification of the execution plan if the initial plan becomes sub-optimal. Three drawbacks exist in this approach: (1) the collection of accurate and correct statistics in a quick way might not be possible, (2) the optimizer might generate execution plans based on uncertain statistics and (3) the loss of the present work up to the point of re-optimization might defect the pipelined execution. In order to overcome those drawbacks, [BAB 05b] proposes a new method of optimization: **proactive re-optimization**. This method incorporates three techniques: (1) the uncertainty in estimates of statistics is computed in the form of bounding boxes around these estimates, (2) these bounding boxes are used to pick plans that are robust to deviations of actual values from their estimates, and (3) accurate measurements of statistics are collected quickly and efficiently during query execution.

The method proposed by [BAB 05b] has similarities with earlier approach of **parametric optimization** of [IOA 92] in which idea is to postpone part of the query optimization until runtime. Typically, this technique chooses a *set* of query plans at query optimization, and identifies a set of conditions that are used to select one of those plans at

runtime. Parametric optimization, along with the choose-plan operator, enables the optimizer to defer the choice of plan to run-time. Switchable plans and switch operators in [BAB 05b] are similar to the choose-plan operator of [IOA 92], however, unlike choose-plan operators, switch operators may occur within pipelines. Furthermore, parametric optimization [IOA 92] does not consider uncertainty in estimates, collection of statistics during execution, robust plans, or re-optimization.

3.2.2 Intra-operator

In this sub-section we want to present two methods of intra-operator adaptive query optimization. Eddy of Telegraph project [AVN 00, BIZ 05 and ZHO 05] and adaptive data partitioning (ADP) of Tukwilla project [IVE 04].

The **Telegraph** project [AVN 00] aims to build a query engine over Web information sources based on an adaptive data-flow paradigm. The objective is to adaptively route unpredictable and bursty data-flows through computing resources. The query processor continuously reorders applications of pipelined operators in a query plan at run-time on a tuple-by-tuple basis. It uses the concept of ***eddy***, defined as a *n*-ary tuple router interposed between *n* data sources and a set of query processing operators. An eddy encapsulates the ordering of operators by dynamically routing tuples through them. The idea is that there are times during the processing of a binary operator (e.g. join, union) when it is possible to modify the order of the inputs without modifying any state in the operator. Such times are called moments of symmetry. They occur at the so called synchronization barriers. For the case of *merge join*, this corresponds to one table-scan waiting until the other table-scan produces values larger than any one seen before. The focus is on join algorithms with frequent times of symmetry, adaptive or non-existent barriers, and minimal ordering constraints. Efficiency of the system depends tightly on the routing policy used in eddies. Different routing policies need to be used under different circumstances. They depend on operator selectivity, operator consumption and production rate, join implementation, and initial delays of input relations.

A recent work presented by [ZHO 05] shows a way of application of eddy in a distributed environment. In the new architecture defined, SWAP, eddy is placed on each execution site. When a query is posed on an execution site, this site becomes the coordinator of the query. In [BIZ 05] an approach called content based routing (CBR) is proposed as an extension to eddies. The eddy processes a query by routing input stream tuples through operators specific to that query where tuples from the same source are routed identically. With CBR tuples are differentiated based on the content and used to route different data to different plans. The idea is to eliminate tuples sooner, reduce latency and improve overall system throughput.

We see an intra-operator adaptive query processing example again in the context of **Tukwilla** project, **adaptive data portioning (ADP)** [IVE 04]. Execution plan of the query is monitored at run time: when a current plan becomes sub-optimal it is replaced by a new plan. Each plan represents a data partition so when an execution plan is replaced, a new partition is dynamically produced. Each execution plan produces a part of the result, the result of the query is found by combining all the results corresponding to partitions. When compared to the approach of eddy which uses local heuristic for routing, this approach is more global in producing new plans.

As we previously indicated, in data integration systems some of the sources have limited query capabilities. These limitations require specialized methods of modeling restrictions, finding possible and feasible plans to answer the query and operators to handle source restrictions. In our next section we will focus on these issues.

4. Query Optimization in the Presence of Restricted Sources

We had already given two examples in chapter 1 about source restrictions. As seen with these two examples, data integration systems have to cope with the different and restricted query interfaces of the underlying data sources. Data sources are restricted, mainly due to two main reasons: (1) interface to the underlying data is actually limited and (2) the source may provide limited access capabilities for reasons of security or performance. First, the integration systems need descriptions of the query capabilities of each source, i.e.; the set

of queries supported by each source. Second, they need special algorithms for the operators in the query execution process. Third, the integration systems need algorithms for deciding how a query can be answered given the capabilities of the sources at a minimum cost. In the following subsections we will discuss the issues and the problems related with those three needs of the integration systems in the presence of restricted sources.

4.1 Abstraction Mechanisms for Describing the Query Capabilities

Capabilities (access restrictions and query processing capabilities) of the data sources in data integration systems are described by using binding patterns [MAN 01, RAJ 95 and YER 99], operator model [HAA 97, PAP 95 and TOM 98], free grammars [DUS 98], ontology based approaches [BAY 97, GRA 97, MEN 98] or recently using web services [SRI 06]. We will explain the first method in detail since our work presented in chapter 3 and 4 of the thesis, is based on the assumption that the heterogeneity of the data sources is abstracted by means of binding patterns.

A binding pattern *bp* for a table $R(X_1, X_2, \ldots, X_n)$ is a mapping from X_i to the alphabet {b,f}, where $1 \leq i \leq n$. The meaning of the binding pattern is as follows; that X_i mapping to *b* are bound; i.e. their values must be supplied in order to obtain data from R, while values of the attributes mapping to *f* are free; i.e. they can be obtained from R when the values of all *b* attributes are supplied.

TSIMMIS [RAJ 95] is the first project in which the heterogeneous data source capabilities are described by means of binding patterns. A data source is described by a conjunctive rule, whose head relation is adorned by a binding pattern. The meaning of the source description is that the data it contains can only be accessed if bindings for the variables bound in the view head are provided. An answer to the query is a semantically equivalent Datalog program, expressed only in terms of the views. In [YER 99] extends the standard binding patterns proposed in [RAJ 95]; more complex system of annotations is presented. Besides *b* and *f* adornments they propose using two extra annotations; constant-*c[s]* and optional-*o[s]* where *c[s]* is a fixed set of constants and its value must be specified in the query, and for *o[s]* its value may or may not be specified. In the Information Manifold, the

content of a data source is specified in terms of hierarchy classes [KIR 95]. A capability description S_{in} and S_{out} is associated to each data source. Similar to bound attributes S_{in} consists of the set of the values of the attributes that must be provided and S_{out} contains the values of the attributes retrieved from the data source, similar to f attribute.

In the following part of this subsection we will give the approach to describe the query capabilities of relational, object and XML data sources using binding patterns. In doing this, we will give a little summary of the part related with describing source capabilities of the work of [MAN 01] that presents the data integration system of LeSelect.

Modeling relational data: In modeling a relational table, we only have to determine the binding patterns of the table. Consider a table R(X, Y, Z) stored in a RDBMS. If full scan of the table is allowed, its binding pattern set should be $R(X^f, Y^f, Z^f)$. However, if an index R.X exists in the RDBMS, a second binding pattern of the form $R(X^b, Y^f, Z^f)$ is used to model the alternative access path.

Modeling programs: The program takes as input a tuple of n arguments $X_1, X_2, \cdots, X_n$ and returns zero or more tuple of arguments of the form $Y_1, Y_2, \cdots, Y_m$. In LeSelect such a program is modeled as a relational table with the following binding pattern $p(X_1^b, X_2^b, \cdots, X_n^b Y_1^f, Y_2^f, \cdots, Y_m^f)$.

Modeling tables with binary attributes: A blob (large data object) A is defined to be binary in LeSelect; if A is mapped to f with some binding pattern of R, then R contains a small sized attribute blobId which identifies A and R has at least one binding pattern $R(blobId^b, A^f)$.

Modeling object data: The idea behind modeling collection of objects with binding patterns is to capture data members of a class by a simple table and the methods of a class as programs. Some object-oriented features like inheritance are not captured by these relational tables. Each class is modeled in isolation, with no link to sub-classes and super-classes. The updates that methods may perform to change the internal state of the object are ignored; in a

sense all the methods are assumed to be read-only. Another restriction is method arguments cannot have collection types etc.

Modeling DOM compatible data sources: Tables with binding patterns can be used to describe any data source that is compliant with the DOM interface. Document Object Model is a generic API standardized by World Wide Web Consortium that enables the manipulation of hierarchically structured data sources. A DOM representation of a data source can be obtained in different ways; i.e. DOM parser provided by IBM [XML 01] to transform an XML or HTML files into tree-structured representation. Generic methodology for representing object data, one table per possible navigational DOM API method is used. *N* denotes the set of all specialized node types identified by the DOM API:

N = {Attribute, CDATASection, CharacterData, Comment, Document, DocumentFragment, Element, Entity, EntityReference, ProcessingInstruction}

Modeling a full-text XML index: An alternative approach to access the textual documents is using full text index (FTI). This offers the following basic functionality: given a keyword, retrieve the elements from several documents that contain that keyword. Such an index is then modeled as the following table:

FTI(word, docID, eIID), with bp(FTI) = {FTI($word^b$ $docID^f$ $eIID^f$)}

4.2 Query Execution in the Presence of Restricted Sources

In [MAN 01] an approach to integrate heterogeneous sources, such as data and programs, in a distributed environment is investigated. The author first creates simple concept based on binding patterns for modeling resources, second presents efficient operators capable of accessing data of restricted sources and finally transfers demonstrates query optimization algorithms in the presence of binding patterns. In this section we will start by a summary of the behavior of these operators.

The presence of access restrictions, formalized using binding patterns, makes the regular set of relational operators insufficient in order to answer queries. So the author in [MAN 02] proposes two new operators: BindJoin and BindAccess, and demonstrates their performance with costly computations and large data transfers. In query execution with these operators caching, inter and intra-operator parallelism techniques in order to reduce total work and early output metrics.

BindJoin operator: The standard relational join operator does not cover well the semantics of combining two tables if at least one of them has access restriction. For example consider a QEP fragment that joins the YP(name^{b} address^{f} phoneNo^{f}) with Employee(name^{f} salary^{f}) table on their name field. Due to the commutativity of the standard join operator we might try to write this fragment as Employee $\bowtie_{name}$YP, or as YP $\bowtie_{name}$Employee. However while the first one is valid, the second one is not due to the binding pattern of YP. We cannot start accesing it before supplying bindings for its name field. So the variant of a relational operator, BindJoin operator (denoted as $\overrightarrow{\bowtie}$) is adopted to capture the asymmetric behavior: the right-hand child of a BindJoin operator cannot be executed on its own, since it depends on the join values passed from the left-hand side of the join operator.

BindAccess operator: Due to the semantics of the binding pattern YP(nameb phoneNof), we cannot perform a scan on YP table. Instead we have to supply some values for the name attribute in order to get YP tuples. Furthermore the set of tuples we can extract from YP, following this binding pattern, depends on the values that we supply for name (in contrast, the result of scan is always the same). To capture the special semantics of a restricted access, we use a special BindAccess operator. It can be thought of as being a "parameterized scan". Where the parameters are the values supplied for the bound attributes.

In Figure 2.1, we see the operation of the BindJoin operator. It receives from its left-hand side tuples of the form (name, salary) and uses name attribute to access the resource YP, following its binding pattern (name^{b} address^{f} phoneNo^{f}). BindAccess operator returns (name, phoneNo) tuples for each name, and the BindJoin cocatanates these tuples with salary attribute.

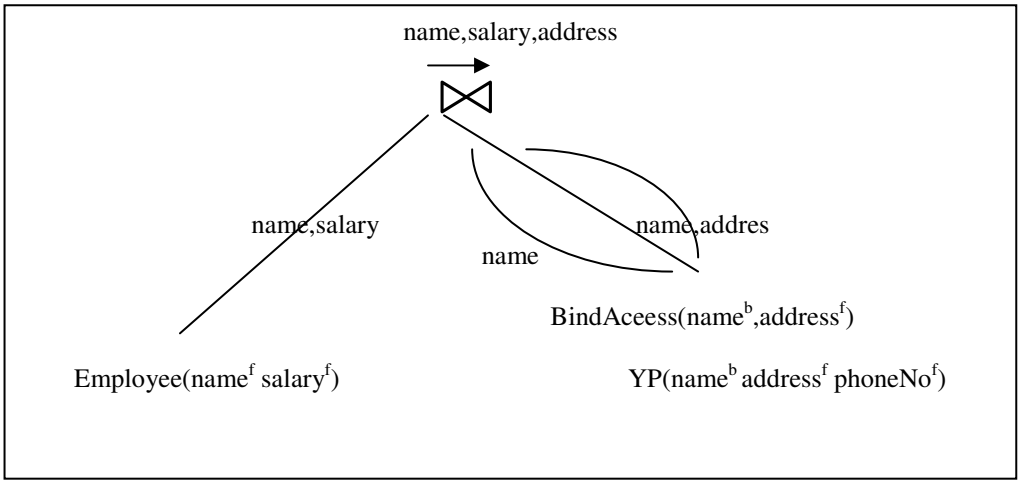

Figure 2.1: BindJoin and BindAccess operators

[GOL 00] is another interesting study for combining the query facilities of traditional databases with existing search engines on the Web. The study basically relies on the dependent joins to supply bindings to the virtual tables when integrating Web searches into a SQL query. WSQ, for Web-Supported (Database) Queries, leverages results from Web searches to enhance SQL queries over a relational database. DSQ, for Database-Supported (Web) Queries, uses information stored in the database to enhance and explain Web searches. The queries supported by WSQ are enabled by two virtual tables, whose tuples represent Web search results generated dynamically during query execution. WSQ query execution may involve many high-latency calls to one or more search engines, during which the query processor is idle. A lightweight technique is used; asynchronous iteration that can be integrated easily into a standard sequential query processor to enable concurrency between query processing and multiple Web search requests.

Figure 2.2 shows the behavior of the dependent join over two tables; Signs(Name) and WebPages(SearchExp, T1, T2, ..., Tn, URL, Rank, Date) where T1, T2, ..., Tn are bound attributes. EVScan in turn contacts WebPages and registers and returns the tuples with T1 = Sigs.Name and T2 = 'Knuth'. For this plan iterator-based execution model is assumed where each operator in the plan tree supports Open, GetNext, and Close operations. The Dependent Join operator requires each GetNext call to its right child to include a binding from its left child. The EVScan operator is an external virtual table scan. Without parallelism, EVScan

performs a sequence of Web searches during execution of this query plan (one for each GetNext call), and the query processor may be idle for a second or more each time. Intuitively, the query processor issues many Web searches simultaneously, without the overhead of a parallel query processor. To achieve this behavior asynchronous iteration is used.

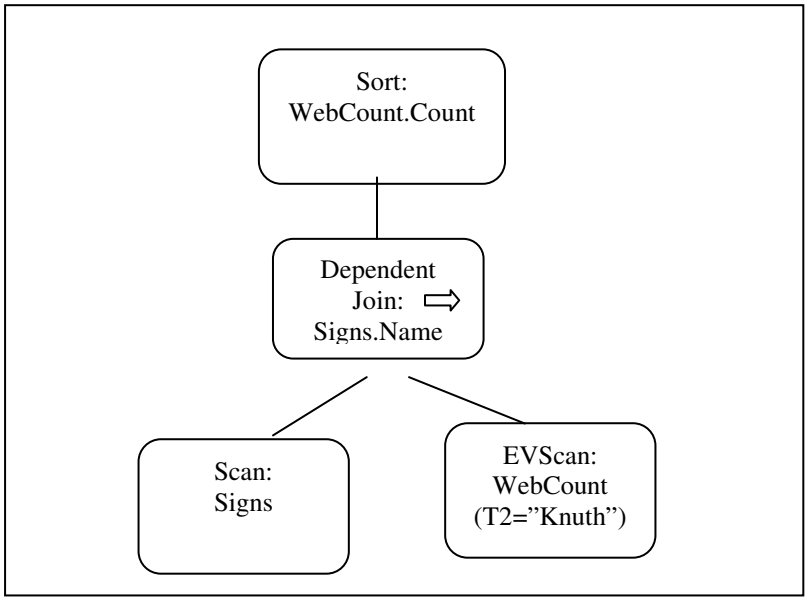

Figure 2.2: Query plan for Signs⋈ WebCount

A very similar idea of the approaches presented above is used in [SRI 06] in optimizing Select-Project-Join queries spanning multiple web services. The main idea is to develop an algorithm for arranging a query's web service calls into a pipelined execution plan that optimally exploits parallelism among web services some of which has access restrictions, to minimize the query's total running time.

4.3 Execution Space Size in the Presence of Restricted Sources

Let us look at a theoretical study given in [MAN 01] on the size of the execution space with binding patterns. The study examines the size on two measures: the number of valid complete query execution plans (plans that can be answerable with present binding patterns of the sources covering all the conjuncts of the query) and the number of valid partial query execution plans (plans that can answerable with binding patterns but does not cover all the

conjuncts) generated by dynamic programming style optimizer. Number of partial plans is considered in order to get prediction about the complexity of the dynamic programming optimization in the presence of binding patterns.

Consider a chain query:

$Ans(u_1,\ldots u_n, v_1,\ldots v_n)$:-$R_1(u_1,v_1),R_2(u_2,v_2),R_3(u_3,v_3),\ldots R_n(u_n,v_n)$

where $v_1 = u_2$ and $v_2 = u_3$ and …. $v_{n-1} = u_n$.

Table 2.2 demonstrates the results of the analysis; where the columns show the number of valid bushy query execution plans, number of valid left-linear query execution plans, number of partial query execution plans and number of partial left-linear query execution plans respectively. So column 2 and 3 indicate the number of valid complete query execution plans whereas columns 4 and 5 show the number of partial query execution plans as an indication of the complexity of dynamic programming optimization algorithm.

In the first line we find the results of the case where there is no binding pattern related with the relations and no selection. It is for comparison purpose, in fact. In the second line the most restrictive case is shown; all the relations have binding patterns. There is only one left-linear solution. There are several solutions with bushy trees but still less than line 1 (classical case). The number of plans considered by dynamic programming also decreases dynamically between line 1 and 2. In the next line refines the analysis, by allowing some of the relations to have ff binding patterns. Exact number of solutions depends on the number of relations having bf as binding pattern. If none has bf the results are equal to the results of line 1, if all bf the results are equal to those of line 2. Line four represents the case where we can start from both ends. All left-linear plans are obtaining by shuffling a join from the left with the ones from the left; there 2^n ways to do this. The complexity is higher than line 2, less than line 1. Line 5 is the illustration of the case where the complexity increases since there are additional access patterns.

So the observations from the table can be summarized as; for some query shapes the presence of binding patterns significantly reduces the number of valid plans. So the size of the execution space for the dynamic programming algorithm of the optimization reduces

significantly. For other cases, size of the execution space is within the exponential complexity of join ordering. However, whether the size of the execution space is reduced or not, the complexity of the algorithm is not proportional to the size of the result execution space but proportional to the partial query execution plans.

Graph	Bushy QEPs	Left-linear QEPs	Bushy partial QEPs	Left-linear partial QEPs.
u free, v free no selections	$\binom{2(n-1)}{n-1}(n-1)!$	$n!$	$3^n - 2^{n+1} + 1$	$n(2^{n-1} - 1)$
u bound, v free selection on x_0	$\binom{2(n-1)}{n-1}\frac{1}{n}$	1	$\frac{n^3-n}{6}$	$n-1$
u free, v free *or* u bound, v free no selections	$\leq \binom{2(n-1)}{n-1}\frac{(n-1)!}{((\frac{n}{m})!)^m}$	$\frac{n!}{((\frac{n}{m})!)^m}$	$\leq \frac{((\frac{n}{m})^3+6(\frac{n}{m})^2+5(\frac{n}{m})+6)^m}{6^m}$	$\leq m(\frac{n}{m}+1)^m$
u bound, v free u free, v bound selections: x_0, x_n	$\binom{2(n-1)}{n-1}\frac{2^n}{n}$	2^n	$O(n^6)$	$O(n^4)$
u bound, v free u free, v free no selections	$\binom{2(n-1)}{n-1}n^n$	n^{n+1}	$O(5.36^n)$	$O(n3.73^n)$

Table 2.2: Bounds on the size of execution space for dynamic programming optimizer in the presence of binding patterns, for a chain query [MAN 01]

4.4 Optimization Methods in the Presence of Restricted Sources

Query optimization in the presence of access restrictions requires algorithms to find stable plans (set of possible plans with the access restrictions of the sources) and at the same time approaches to find optimum execution plans in terms of the cost. Some approaches for optimizing queries in data integration use strategies to find query execution plans when limited access patterns are available on data integration [MOR 88 and RAJ 95]. In these works they ask the question of whether a there exists some ordering of accesses to the data sources to answer the given query. They do not focus on the question of finding an optimal order. On the other hand recent works focus on both questions optimizing the coverage and

cost; either in a decoupled way-attempting to optimize coverage and cost in two separate phases [MAN 01 and LI 03] or in a combined manner to handle two phases together [NIE 01 and LAM 04].

In [MAN 01] two optimization algorithms are proposed; (1) dynProg- an extension to the traditional System-R optimization algorithm [SEL 79], based on dynamic programming that handles binding pattern restrictions and (2) bestFirst- improving the first algorithm in a way to produce first plan relatively quickly. New pruning rules are presented to allow combining selections, joins and BindJoins (similar to regular joins with a difference of access restrictions on the right-hand site of the operator) at every step in optimization process, in order to try all interesting placements of selections in the QEP. As a last attempt regarding query optimization the author proposes a distributed optimization approach with rules for operator localizations.

[LI 03] studies the problem of generating efficient, equivalent rewritings using views to compute the answer to a query taking the closed-world assumption, in which views are materialized from base relations, rather than views describing sources in terms of abstract predicates, as is common when the open-world assumption is used. Query optimizers take a logical plan (a rewriting of the query) as an input, and generate efficient physical plans to compute the answer. Thus the goal is to generate a small subset of the possible logical plans without missing an optimal physical plan. They first consider a cost model that counts the number of sub-goals in a physical plan, and show an execution space that is guaranteed to include an optimal rewriting, if the query has a rewriting in terms of the views. They also develop an efficient algorithm for finding rewritings with the minimum number of sub-goals. They then consider a cost model that counts the sizes of intermediate relations of a physical plan, without dropping any attributes, and give an execution space for finding optimal rewritings.

In [NIE 01] they present techniques for joint optimization of cost and coverage of the query plans. Their algorithms search in the space of parallel query plans that support multiple sources for each sub-goal conjunct. The refinement of the partial plans takes into account the potential parallelism between source calls, and the binding compatibilities between the

sources included in the plan. They briefly review how to compute the cost and coverage of a parallel plan. Next, they provide both a System-R style query optimization algorithm as well as a greedy local search algorithm for searching in the space of such query plans.

5. Discussion on Query Optimization Methods in Large Scale Data Integration

So far we have introduced the need and characteristics of the environment of data integration in large scale. In order to summarize; we can state that many of today's applications require access to data residing on different and distributed sources. So, initial problem is to find means to integrate them in a way to provide uniform and transparent access to users. Theory behind this integration and main approaches for integration are explained in chapter 1. When it comes to query optimization; we face challenges like scalability, autonomy preservation, optimization performance, adaptivity and the presence of restricted sources.

Main focus of some of the research on data integration systems is based on extending the classical cost model in order to capture information from autonomous data sources and make better estimation about the cost of the query. However, dilemma for such systems relates to getting optimization information (e.g., statistics, cost model) from autonomous sources and handling the cases with unpredictable events in query execution time. This need leads way to adaptive query optimization. As presented in section 2 of this chapter there is tremendous research on adaptive query optimization behind which the main idea is to optimize at compile time and in run time if variations from compile time estimations exist or run time conditions require make re-optimization. We want to focus on the point of decision making about the necessity of re-optimization. As clearly explained by the work [HUS 06] who makes this decision is critical in large scale network since network latency is high and bandwidth is low. This faces us with the challenge of minimizing the amount of communication in order not to cause bottleneck on the optimizer's site. If the monitoring of the run-time conditions and making the decision about adaptivity is done by a central authority, it is most probable that there will be bottleneck on the optimizer's site due to amount of necessary message passing towards a single point. This argument increases the role

of recent decentralized approaches like presented in [ARC 04, COL 04, JON 03, IVE 99, MOR 03, OZA 05a and URH 00]. Double hash join of TUKWILLA [IVE 99] and Xjoin of [URH 00] are classified in [HUS 06] as “decentralized dynamic operators” whereas execution model based on communicating mobile agents [JON 03] and execution model based on brokers [COL 04] are classified as “decentralized dynamic optimization methods at the sub-query level”.

TUKWILLA [IVE 99] that is already presented in section 2.2.2.1 of this chapter is an example of data integration project outstanding with its decentralized re-optimization style since the one of the adaptations imposed by it is done within the execution engine without invoking optimization at run time. Double hash join operator presented in this context maintains two separate hash tables at the same time in contrast to conventional hash join which creates the hash table from the smallest operand and then probes it with the tuples of the second operand. This technique aims to minimize the time of production of the first tuple introduced, to eliminate the need for the optimizer to know the sizes of the operands in order to choose the operand to be used for the production of the single hash table and to mask the slow arrival rate of the tuples from corresponding operand.

The double hash join can lead to bad performance due to memory limitations in handling multiple hash tables and to possibility of the cases where the productions of the two operands are blocked. So, XJoin [URH 00] is introduced which is similar to double hash join but it has an additional phase which is in charge in the cases of memory limitation and blockage of the arrival in receiving data from the operands. This additional phase is responsible for joining the tuples present in the memory with the tuples residing on the secondary storage in parallel with the execution of the double hash join. The two methods [IVE 99 and URH 00] focus on intra-operator level adaptivity whereas [COL 04 and JON 03] propose adaptive query optimization methods at sub-query level.

Execution model based on brokers presented in [COL 04] aims to provide flexible, dynamic and decentralized execution of queries submitted to the data integration system over large scale network. The broker is the basic unit responsible from the execution of the sub-query. It detects the inaccuracies and adapts accordingly. In doing this it communicates with

the user and the other brokers to notice the changes in the execution environment. Although this method is an example of decentralized and sub-query level of adaptivity; its focus is on improving the cost of local processing. Another dynamic and decentralized execution model is proposed in [JON 03], it is based on communication mobile agents. In this model each mobile agent is in charge of executing a sub-query of an execution plan and can modify its plan for the remainder of the execution in the cases of inaccuracies. Each agent communicates with other agents of the query and compares partial results for the remainder of its execution plan. In this method, the amount of control messages generated with increased number of agents might generate congestion of network.

All the decentralized methods explained above improve the local processing cost with their decentralized decision making about adapting the use of CPU, I/O and memory in the presence of changes in the execution environment (e.g. estimation errors, delays in data arrival rate, etc.) but they do not take into account the necessary concern for reducing the amount of the volume of transferred data over large scale network. [OZA 05a] shows that transfer cost can reach up to 80-85% of the foreseen total cost of the query.

At this point, [ARC 04] proposed a solution which is based on mobile join that is capable of making its decision about changing its place of execution in the cases of variation between compile-time estimations of the optimizer and run time conditions (Table 2.1). In this model mobile join decides the site on which it should continue its execution based on the embedded cost model proposed in [HUS 05] and the cooperation methods given in [MOR 03], in a distributed, autonomous and decentralized way. The aim of each operator, in its reaction to estimation errors or run-time conditions, is to minimize the volume of data transferred. Here we end our discussion for a while and focus on mobile execution model in more detail in next section.

6. Mobile Query Execution Model in Large Scale Data Integration

In this section we will present mobile query execution model starting by a summary of the environment. In the next subsection, we will present the algorithm of mobile hash join operator and performance evaluation between the mobile join and standard join with respect

to changing estimation error on the size of the data and network bandwidth. In sub-section 6.2, we will explain the way of interactions carried out by the mobile join operator with the mediator and with the participating mobile operators of the query.

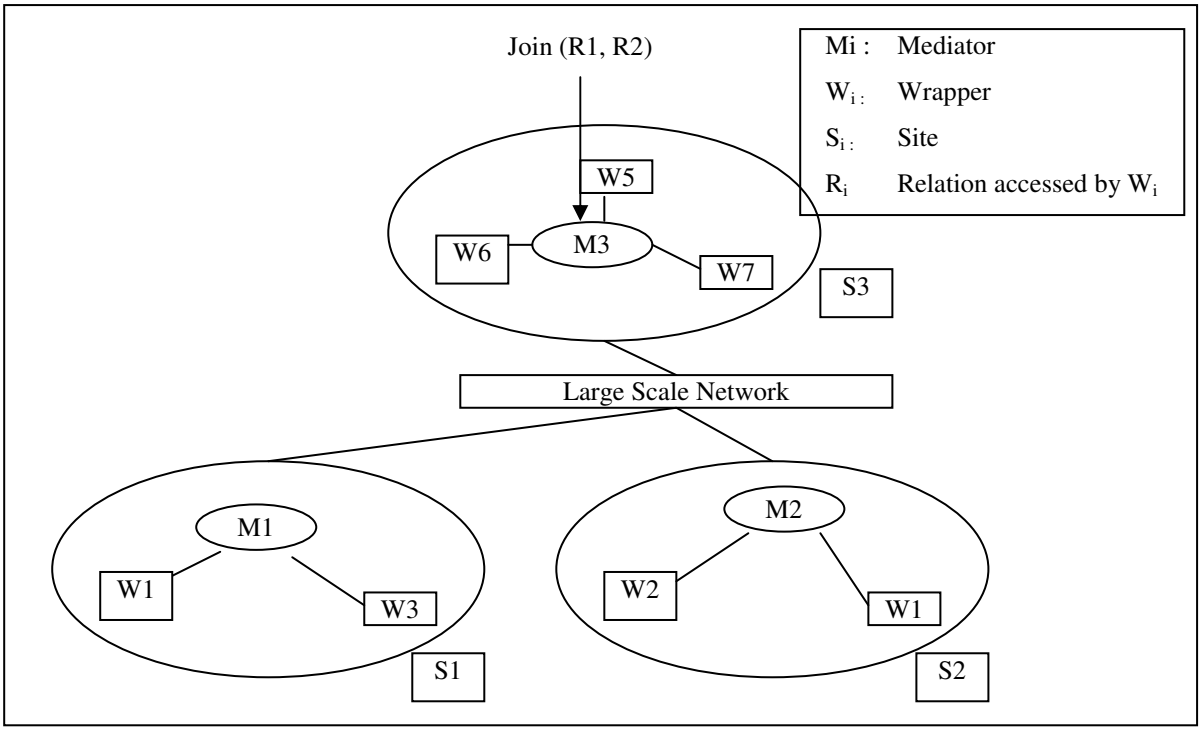

Figure 2.3: Environment of Mobile Query Execution Model [HUS 05]

On each site participating in the query evaluation, there is a mediator as presented in [HUS 05]. The query is submitted to the mediator where it is transformed and optimized (Figure 2.3). The execution plan consists of set of relational operators. The operators of an execution plan are executed either by the mediators or by the wrappers. Every operator of the plan is executed by a mobile relational operator on the mediators and by a fixed operator on wrappers. On the wrapper site, the translation of the query and re-formatting of the result for the mediator will be abstracted by means of a scan operation.

6.1 Mobile Join Operator

Now we will present briefly the algorithm of a mobile join operator based on simple hash join [ARC 04]. Let us consider the example to illustrate the mobile execution model; R1 and R2 are base or temporary relations, with R1 located on site S_1 and R2 located on site S_2

and |R1| < |R2| (|R1| is the size of R1); T is the result of join between R1 and R2, T ← Join(R1,R2), which must be materialized on site S_1 where T is expected. In hash join, in first step, the hash table is built from the smallest relation R1; in a second step the hash table is probed with the other relation R2. During the building of the hash table, statistics on R1 can be computed precisely from the contents of R1. Then from precise statistics on R1 and compile-time statistics on R2, the selectivity factor for example can be revised and statistics on T consequently can be refreshed. Then the mobile join operator decides on its own whether to move or not (on S_2 or on another site) by means of decision function that computes the migration site. The decision function is based on a cost model [HUS 05] extended with cost of mobility and have parameters StatInfo (updated statistics on relations involved), DataAvailability (waiting for the first tuple and waiting time for any tuple) and SystemState (communication bandwidth, CPU, memory capacity). Hash table is moved along with the mobile join operator. For example T is expected on S_1 and compile-time estimate of the optimizer is |R2| < |R1| + |T|: consequently the mobile join operator is placed on S_1; however if it appears that |R2| > |R1| + |T| after building the hash table on S_1 the mobile join operator migrates to site S_2 for probing. The algorithm given in Figure 2.4 illustrates the behavior of the mobile join operator.

```
Step 1: If(not local(R1)) then receive(R1);
        Build(R1, HT1);                                  //and compute the statistics on R1
        Site ← Decision(StatisticalInformation, DataAvailibility, SystemState);
         if (not local(Site)) then migrate on Site;

Step 2:  if (not local(R2)) then receive (R2);
        Probe(HT1, R2, T);
        if (not local(T)) then send(T) else materialize(T);
```

Figure 2.4: Algorithm of a mobile hash join operator [ARC04]

In [OZA 05a] comparative results of performance evaluation between the mobile join and standard join are given in detail. This performance study shows the efficiency of mobile execution with respect to the accuracy of the statistics; the mobility is effective (i) only from

30 % of errors (over or under-estimation) on the estimation of the size of the relation R1 made by an optimizer and (ii) when the network bandwidth is weak (640 Kb/s - 1.28 Mb/s). In large-scale data integration where data sources are expected to be heterogeneous and autonomous; the various data sources do not export needed statistical information or the statistical information is subject to be obsolete. It is difficult for an optimizer to estimate the cost, thus 30 % of error in the estimation is not high. In addition, a network bandwidth of 640 Kb/s can often be experienced. So in this context, the contribution of mobility can be taken into consideration.

6.2 Interactions of the Mobile Join Operator

In terms of the interaction of the mobile join operator in query execution; we want to discuss about two works [HUS 05 and MOR 03] where the first one is focusing on describing the cost model within different components of the mobile query execution model; mediator, mobile join operator and the wrappers. Together with the cost model, different interactions of the mobile join operator with the mediator are defined as well. [MOR 03] proposes three cooperation methods describing the control and data communication between the mobile agents of the query.

In the work presented in [HUS 05] a cost model which is composed of two parts is proposed; first part resides on the mediator and the second part is embedded in the mobile agent. Mediator cost is used to evaluate the cost of the execution plan of the query submitted to it whereas, the mediator cost model and the embedded cost of the agent are used to estimate agent's metrics. The cost model of the mediator is composed of the profiles of the data sources known by it, characteristics of the architectures of the mediators interacting with it, the processing cost of the wrappers and their cost formulas. First three of this information is stored in the catalogue with a time stamp of last update. In case the formulas of the wrappers are not exported to the mediator, the mediator will use generic formulas to estimate the response time of the wrappers related with its operations. The agent cost model is supplied to the agent during query optimization. The agent cost model is composed of its migration space (set of sites the mobile agent can migrate), the profile of the second operator and the parameters of the cost estimating the production of the second operand.

The agent interacts with the mediator at two times; during its installation phase and its decision for migration phase [HUS 05]. When it installs on the mediator it asks for the information on the workloads of the sites in it migration space, the replication sites of the second operand within its migration space and the bandwidth of the interconnections between the mediator and the replication sites of the second operand existing in its migration space. While the agent is building its hash table from the first operand, the mediator prepares required information. During its decision phase agent asks the mediator to calculate the profile of the result relation based on the profile of the first operand discovered at run-time and the profile of the second operand present in the agent with a time stamp so that mediator can choose the latest information (between the information it knows and the information given by the agent). The second information asked by the agent is the estimated execution time of its execution on each site present in its migration space.

Performance evaluation of the work presented in [HUS 05] shows the agent chooses the best site to continue its execution whatever the experimentation environment is (local or large scale). In large scale environment local processing cost is almost negligible when compared with the transfer cost of data so it is proper to say that the agent must choose its site only according to the transfer cost. On the other hand, in local environment the migration decision should take into account workload of the execution site together with the transfer cost since workload of the site can decrease the response time up to 22 %.

In order to end our explanation of mobile query execution model; we want to also look at the cooperation methods of mobile relational operators presented in [MOR 03] since we find this important in terms of extending the strength of mobile query execution model to sub-query level. These cooperation methods and the cost model presented previously are complimentary in fact. Here, the tree structure of an execution plan remains unchanged at run time but the localizations of the operators in the execution plan are changed dynamically. Three cooperation methods are proposed in this perspective between the mobile join agents in a large scale distributed database. These methods allow a mobile agent to make its decision to migrate or not according to the decisions of the other communicating agents with it. The

differences between the methods are: (i) propagation or not, of the estimation corrections, and (ii) cooperation or not, before the join execution.

7. Conclusion

In section 2 of this chapter we have discussed issues related with adaptive query optimization on an axis of centralized vs. decentralized decision making for adaptivity. As to remind we can state once more that decentralized decision making is critical in order prevent the bottleneck on the optimizer's site, due to number of required message passing on large scale network. We also have outlined several decentralized optimization efforts which delegates the decision of adaptivity to operators or execution units of sub-query. However the focus of these works was not reducing the volume of transferred data during query execution which we think is one of the most critical problems over large scale network with high network latency and low bandwidth. With these arguments mobile query execution model presented in section 5 is worth considering since the way of adaptivity of mobile relational operators is migrating from one site to another if they discover errors on the compile time estimates of the optimizer, in order to reduce the amount of data volume transferred.

However mobile query execution model does not capture the needs of the sources with limited query capabilities. All the previous work on this model is based on the query execution with mobile relational operators capable to handle positive queries (queries posed to the sources with no access restrictions). Yet, this is not always the case in data integration since some of the sources have limited query capabilities as we explained previously, due to performance or security reasons. When querying such sources we cannot retrieve all possible information due to source restrictions. From integration point of view the restrictions imposed by these sources should be abstracted. Although there are several methods; "binding patterns" is widely used, where the attributes of the source are represented together with its "free" or "bound" characteristics. So in order to retrieve data of the free attribute, the values corresponding to bound attribute should be given. In query optimization, mechanisms to find the valid plans with defined restrictions are critical together with finding optimum plans. The effort of finding valid plans leads decrease in the size of the execution space and increase in the complexity of these algorithms. And finally in query execution specific versions of scan

and join operators are required in order to handle parameterized scan and asynchronous join type of operations. It is correct to state that all the efforts presented so far on mobile query execution model, do not take into account the requirements of restricted sources in query execution.

Another open problem related with the mobile query execution model is in the initial placement of the mobile relational operators by the optimizer. All previous work assumes that the optimizer locates the mobile relational query operator to a site which minimizes the cost of query execution. But the chosen site might disable the reaction of the operator (in terms of migrating to another site). Mobile query execution model needs links with the optimizer in terms of the placement of the operators on the execution sites since mobile relational operators are sensitive to their initial placement. Initial placement of mobile relational operators at compile should capture needs of adaptivity at run time. So our motivation is to propose solutions for two open problems related with mobile query execution model: “mobile execution in the presence of restricted sources” and “initial placement of mobile relational operators”. In initial placement we try to find an initial site which provides close to optimum performance within an interval instead of an optimum performance on single point estimations.

In the following two chapters we will be explaining two works in the context of mobile query execution model. First work aims to present a mobile query execution model for restricted sources whereas the second work proposes an initial placement approach for mobile relational operators that takes into consideration the possibility of their migration.

Chapter III Mobile Query Execution Model: Mobile Relational Operators for Restricted Sources and Initial Placement of Mobile Relational Operators

1. Introduction

So far, we have presented the context and a state of the art positioning our motivation for the studies that we will present in this chapter. In order to give a summary we can state that;

1) sources in data integration are distributed, heterogeneous, autonomous,

2) adaptive query optimization plays important role since the estimations capturing the data is difficult and unpredictable events can occur during execution time,

3) network latency is high and communication bandwidth is low so central adaptation methods are inapplicable in large scale network,

4) some of the data sources are restricted due to privacy and performance reasons and require specialized modeling, optimization and query execution methods,

5) mobile query execution model, based on mobile query operators in query execution, over large scale network is promising with decentralized decision making about adaptation since it prevents bottleneck due to high message passing on optimizer site, and its adaptation is in a way to reduce the amount of data volume transferred,

6) previous work on mobile query execution model do not capture the needs of restricted sources and

7) previous work on mobile query execution model assumes that the mobile relational operators are placed on the sites with single point estimates of the optimizer and this kind of placement might obstruct the mobile relational operator to move during its execution.

The points highlighted by 6 and 7 of the above statement point out our motivations behind the works that we will explain in this and next chapter. Taking into account the aspects of query processing in data integration systems and the open issues explained so far we want to introduce (1) mobile join operators which are able to handle sources that have limited query capabilities [OZA 05b] and (2) to have link with the optimizer in terms of mapping the mobile relational operators to the sites regarding their sensitivity to their initial placement [ERG 07].

In (1) our challenge is to propose mobile join operators that are able to work in the presence of access restrictions imposed by participating data sources. We designed Mobile Dependent Join (MDJoin) and Sampling Dependent Join (SMDJoin): two join operators which are mobile (in case of inaccurate estimations of compile-time, they adapt at run time by changing their place of execution) and able to work with access restrictions imposed by the sources. The difference between the two new query operators lies in their level of adaptation ability to the execution environment.

In (2), we address the problem of initial placement taking into account the adaptation capability of mobile relational operators at run-time (mobile join, MDJoin and SMDJoin). Our challenge is not finding good placement based on single-point estimates but defining acceptable placement for an estimation interval in order to avoid dramatic performance at run-time.

The rest of this chapter is organized as follows; in section 2, we will present our mobile relational operators designed for restricted sources. Afterwards, in section 3 we will explain the need for robust initial placement strategy in the context of mobile execution providing acceptable performance in the cases of estimation errors. Conclusion remarks will be included in section 4.

2. Mobile Relational Operators for Restricted Sources

In this section we will start by introducing the basic operators required for restricted sources then continue by presenting two mobile join operators designed for restricted sources.

2.1 Basic Operators for Restricted Sources

Suppose we have the following example as seen in Figure 3.1; there are two tables holding data related to the telephone subscribers; ***Telephone(name, telNo)***, ***Address(telNo, address)***. We assume that this database is distributed which means Telephone table is located on a site different than the location of the Address table and the tables are not fragmented. We also assume that those two tables have 'binding patterns' as introduced in sub-section 2.4.2. For those attributes mapped to 'b', the values should be supplied in order to get information from R while the attributes mapping to 'f' do not require any input in order to return tuples from R. If all the attributes of R are mapped to 'f' then it is possible to get all the tuples of R without any restriction (e.g. with a scan operator).

Telephone

name	*telNo*
Ahmet Hoşgör	90-232-8990786
Mehmet Yılmaz	90-312-7645673
Hüseyin Kaçar	90-212-8978990
Leyla Korukçu	90-232-7506530

Address

telNo	*address*
90-232-8990786	234 sok. 31 Bostanlı IZMIR
90-312-7645673	556 sok. 45 Çankaya ANKARA
90-212-8978990	789 sok. 43 Etiler ISTANBUL
90-232-7506530	786 sok. 1 IZMIR

Figure 3.1: Sample database

If we know that the binding patterns of the tables of our sample database are as follows ***Telephone(name^f, telNo^f)***, ***Address(telNo^b, address^f)***, it means that the first table is ready to return the values of the ***telNo*** and the ***name*** of the subscriber without any restriction while the second table can give the ***address*** only if the value of the ***telNo*** attribute is known.

In case of a need to execute a simple query, joining data of two tables on their common attribute telNo we see that regular set of relational operators are insufficient in order to get the answer. The semantics of the scan operator is to retrieve all data from a table however in the case of the Address table, this approach is inapplicable. We need a new operator in a way like parameterized scan which supplies bound attributes in order to retrieve tuples from the table. Similarly, if we look at the semantics of the join operator, it assumes the presence of scan operator on both sides, meaning that the synchronous data retrieval is possible from both sides. But in the case of Address table, this approach is not suitable. Therefore in the following subsections we introduce two operators required in the presence of restricted sources: dependent access (DAccess) and dependent join (DJoin) which are variants of scan and join operators of relational query optimization, designed for asynchronous data retrieval.

2.1.1 Dependent Access Operator: DAccess

Although we can model the restricted sources with formalization of 'binding patterns', due to the access restrictions of the sources we cannot use the query processing operators, like scan and relational join as explained in the previous section. In the example, we have binding patterns of the tables as: Telephone($name^f$, $telNo^f$) and Address($telNo^b$, $address^f$). Binding pattern of the table Address prevents the use of scan operator in order to get the address list of the subscribers. In order to get the address value of the subscriber we have to give the values of the telNo attribute. So we need a new scan operator which is able to deal with the restricted sources in the presence of the binding patterns; we name this operator in our study DAccess as D indicates its dependency on the values of the input attribute(s). While the relational scan operator always returns the same result set from a single relation, this new operator DAccess returns different sets depending on its input set. For example for the value of the telNo attribute given as "90-212-8978890" it returns the tuple ("90-212-8978890", "789 sok. 43 Etiler ISTANBUL"). As seen in Figure 3.2 it is a simple operator which accepts a set as its input and returns another set as its output. 'Formal semantics of DAccess is as follows: Consider a table R(X, Y) where X and Y are disjoint sets of variables and R has a binding pattern given as $R(X^b, Y^f)$. Then denoting $DAccess(R(X^b, Y^f), \chi) = \sigma_{X \in \chi} R(X, Y)$ [MAN 02].

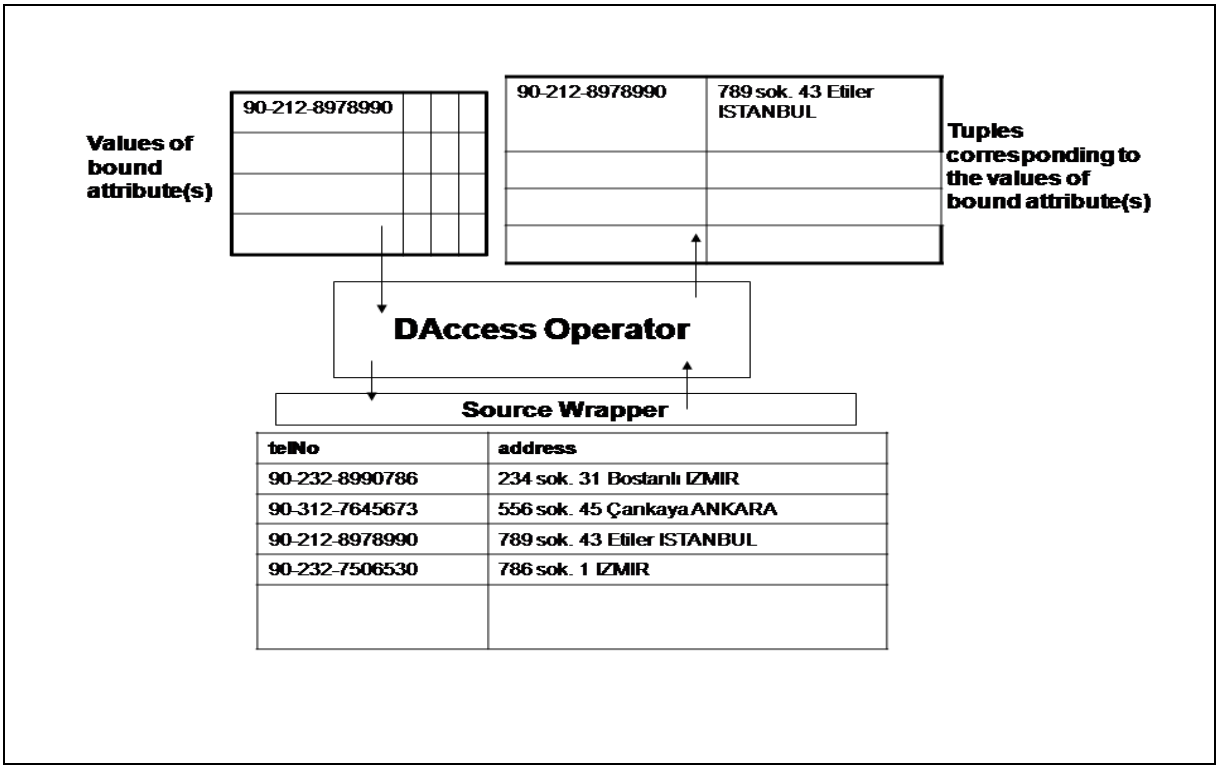

Figure 3.2: Function of DAccess operator

DAccess is a simple operator which is in contact with the wrapper of the restricted source and functions as an adapter to hide the heterogeneity of the query capability of the restricted source. It provides bound attribute(s) and retrieves the corresponding tuples from the source. The wrapper of the source should have corresponding interface to pass the values of the bound attribute(s) to the restricted source and corresponding tuples to the DAccess operator.

2.1.2 Dependent Join Operator: DJoin

In this section we will describe a join algorithm designed for restricted sources and based on the hash join algorithm [SCH 90] where hash table is created from the smaller relation, R1 (building relation) and hash table is probed with the other relation, R2 (probing relation).

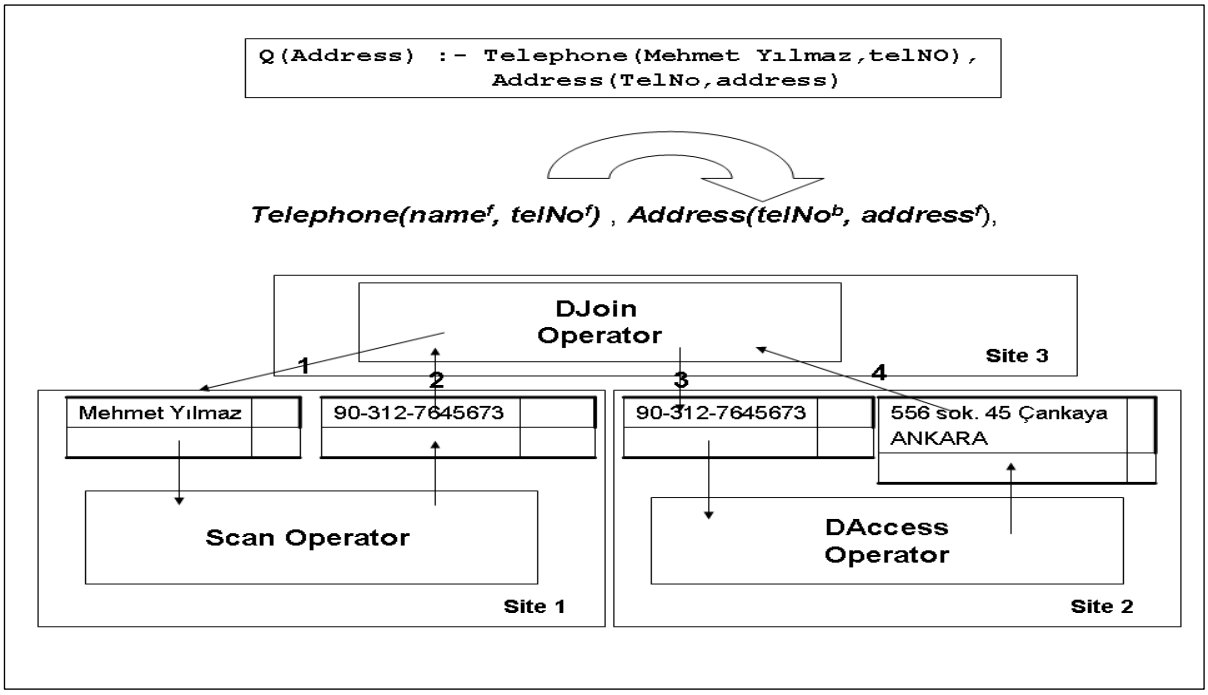

Figure 3.3: Co-ordination of DJoin and DAccess operators

Consider an execution plan composed of a join operation: Telephone $\bowtie_{telNo}$ Address like in the Figure 3.3. In relational join, both of the following fragments; (Telephone $\bowtie_{telNo}$ Address) and (Address $\bowtie_{telNo}$ Telephone) are valid since join is commutative. But with restricted source like Address relation, (Address $\bowtie_{telNo}$ Telephone) is not valid since the binding pattern of the table Address requires the value of the telNo attribute as input in order to return the value of the address attribute. So we need a new join operator which is asymmetric in nature, also known as dependent join [GOL 00], represented by the symbol $\overrightarrow{\bowtie}$ and which cannot be executed on its own, since the left hand side of the join requires values to be passed from the execution of the right hand side. Therefore, we designed a dependent join operator which is able to deal with the restricted sources by cooperating with DAccess operator in internal value passing between the sides, and named it as DJoin. Figure 3.3 demonstrates the way of coordination between DAccess and DJoin operators. Formal representation of the dependent join is $T \leftarrow Scan(R1(U^f, V^f)) \overrightarrow{\bowtie}_{V=X} DAccess(R2(X^b, Y^f)) = \{(u, v, x, y) \mid (u, v) \in R1 \wedge (x, y) \in R2 \wedge u = x\}$.

The algorithm of the dependent join, DJoin is presented in Figure 3.4. The join operator has the relations to be joined (R1 and R2) and the relation where the result of the join operation should be stored is T. V and X represent two sets of join attributes from two relations where X contains the bound attributes of R2 which require value passing from V of R1.

```
DJoin (R1, R2, T: Relation; V, X: Set of Attributes)
    {
    If (not local (R1)) then receive (R1);
    Build (R1, HT1);
    P ← Project (R1, V);

    If (not local (R2))     then
                            {
                            send P to S2;
                            //  DAccess runs on the remote site
                            receive R2' from S2;
                            }
                             else DAccess(R2, P, R2');

    Probe (HT1, R2', T);
    If (not local (T)) then materialize (T) else send (T);
    }
```

Figure 3.4: Algorithm of a Dependent Join (DJoin)

If the first relation is not on the site of the DJoin operator, it is retrieved by a scan operator since we know that the first relation has an ff binding pattern. We build a hash table from the first relation (HT1) and at the same time we retrieve the distinct values of the attribute(s) V and store them into the table P. If the join operation is not on the site of R2 (restricted source), we send P to the DAccess operator of the site S_2. DAccess operator of site S_2 returns the corresponding tuples of R2, denoted as R2'. After receiving R2' the join operator probes the hash table with R2' and stores the result relation in T. If the result site of the query is the site of the join operation, T is materialized or sent to the result site.

We assume that on the site of each restricted source there is a DAccess(R, P, R') where R is the source relation, P is the set of the bound attributes and R' is the set of the tuples of R corresponding to P. This specific operator receives set of values corresponding to the bound attribute of the source as input (P), feeding this set to the wrapper of the source retrieves the data corresponding to the free attributes of the source beneath and stores the result in R'.

2.2 Mobile Join Execution Model for Restricted Sources

An important issue in data integration is the use of adaptive optimization methods as discussed in detail, in chapter 2. In terms of adaptation, these methods try to change the execution plan at run-time, re-optimize query, or use specific operators to deal more flexibly with unpredictable events. Their focus is to face the challenge of optimization with autonomous sources and unforeseen run-time conditions. Although there are many adaptive query optimization methods which are elegantly designed with different adaptation types (e.g. re-optimization, re-scheduling, replacement and dynamic operators), from the point of control, decentralized adaptive query optimization methods are applicable over large scale network. Decentralized query optimization methods do not require a central authority for monitoring and decision making about adaptation where the control is delegated to operator or sub-query level. This type of adaptation reduces the amount of control and data message passing towards a single site over large scale network where network latency is high and network bandwidth is low.

An adaptive query execution model presented by [ARC 04] is one of the examples of decentralized adaptive query optimization. Within the decentralized methods which try to improve the local processing cost (e.g. CPU, I/O, memory) in the presence of changes in the execution environment (e.g. estimation errors, delays in data arrival rate, etc.), the advantage of this execution model comes with its adaptation style as to reduce the amount of the volume of transferred data over large scale network. The execution model of [ARC 04] is based on mobile join operator that is capable of making its decision about changing its place of execution in the cases of variation between compile-time estimations of the optimizer and run time conditions. In mobile query execution model, mobile join decides the site on which it

should continue its execution in a distributed, autonomous and decentralized way. The aim of each operator, in its reaction to estimation errors or run-time conditions, is to minimize the volume of data transferred. However, all the work presented so far in the context of mobile query execution model is based on the relational operators applicable to the sources which do not expose access restrictions. This handicaps the use of mobile query execution model in large scale data integration where some of the sources have access restrictions. Therefore our challenge in the following sub-sections is to present two mobile join operators designed for restricted sources.

2.2.1 Mobile Dependent Join Operator: MDJoin

We want to use the similar approach as explained in section 2.6.1 in designing mobile join operators for restricted sources. As in the case of DJoin, mobile join algorithm designed for restricted sources is based on the hash join algorithm [SCH 90]. The algorithm of our first mobile dependent join operator, MDJoin, is presented in Figure 3.5. Example is the same as the one of DJoin explained in section 2.1.2. If the first relation is not on the site of the DJoin operator, it is retrieved by a scan operator since we know that the first relation has an ff binding pattern. It builds the hash table from the first relation (HT1). By relational project operation it retrieves the distinct values of the attribute(s) V of R1 and store them in the table P just like before. But the difference from static DJoin starts here; after building the hash table the mobile join operator is now able to calculate true statistics of R1 and P and it has better idea about T. MDJoin can make a decision about where to continue its execution and migrates to the Site according to its decision function (i.e. size of R2' + size of P > size of R1 + size of T + size of P).

```
MDJoin (R1, R2, T: Relation; V, X: Set of Attributes)
{
    If (not local (R1)) then receive (R1);
    Build (R1, HT1);                              // compute statistics on R1
    P ← Project (R1, V);

    Site ← Decision (StatInfo,
                     DataAvailabilitiy,
                     SystemState);
    If (not local (Site) then migrate on Site;    // with P and HT1

    If (not local (R2))  then
        {
            send P to S2;
            //  DAccess runs on the remote site
            receive R2' from S2;
        }
        else
             DAccess(R2, P, R2');

    Probe (HT1, R2', T);
    If (not local (T)) then send (T) else materialize (T);
}
```

Figure 3.5: Algorithm of a Mobile Dependent Join (MDJoin)

After migration step it is ready to get data from R2; if it is not on the site of R2 then it sends P to the site S_2 and receives tuples of R2 corresponding to P in R2' through the DAccess operator of the site S_2. In 'Probe (HT1, R2', T)' step, the join operator probes the hash table with R2', and with the last line of the algorithm the result relation is either send to the client site or materialized.

It resembles the mobile join algorithm, in a way having an initial projection step on R1. But the difference between the two operators is it is not possible to probe hash table build from P with R2 like in mobile semi-join. We need DAccess(R2, P, R2') operator on the second source due to the restriction imposed by the source.

2.2.2 Sampling Mobile Dependent Join Operator: SMDJoin

Although the MDJoin operator, with its mobile and autonomous nature, is capable of joining data retrieved from bound relations and it is reactive to the changes in run time environment when compared to the DJoin operator of the standard strategy. All the information it has about R2 is coming from the optimizer which is compile-time information. We can say that it is blind to the run-time information like data characteristics, data availability and system status concerning site S_2. For the cases where the information about the second site is limited or not available, MDJoin can make an improper decision about its location for probing step. To cover such cases we propose to include a mechanism to collect information about second site so as to allow the operator make better estimation about the location of the join operation. Our approach can be seen as an extension of the work presented in [KHA 00] where a probing mechanism is proposed to make better estimate in the presence of possible estimation errors. However the use of probing is quite different in our case: SMDJoin uses probing in adaptation at run time whereas probing is used to have statistics about the data and to choose the optimum execution plan at compile time in [KHA 00].

SMDJoin operator, as shown in Figure 3.6, has a sampling step after building the hash table and projecting (V) from R1 which is stored in P. It has a sampling part (between the points P1 and P2) which consists of selecting a sample of data from P (we use 'p' to name it), sending it to DAccess operator of the site S_2 and receiving $R2'_p$ (R2' corresponding to p). At the point **P1** based on the sizes of P and p, it is possible to retrieve sample ratio given as:

Sample Ratio = Size of P / Size of p.

After completing the sampling step, at point **P2**, we also know the size of $R2'_p$, cost of sending p, cost of computing R2'p and receiving $R2'_p$. So the estimations for the size of R2', costs of sending P, preparing R2' and receiving R2' can be revised leaning on the new information as shown below:

Size of R2' = (Size of R2'p × Sample Ratio) – Size of R2'p

Cost of Sending (P-p) = (Cost of Sending p × Sample Ratio) – Cost of Sending p

Cost of Computing R2' = (Cost of Computing R2'p × Sample Ratio) – Cost of Computing R2'p

Cost of Receiving R2' = (Cost of Receiving R2'p × Sample Ratio) – Cost of Receiving R2'p.

So at the point P2, SMDJoin can choose the site of the join execution with enriched information which includes computed size of P and p, revised size R2', revised cost of sending P, preparing R2', receiving R2'. Remaining part of the algorithm of SMDJoin is quite similar to the one of MDJoin. While receiving data from R2; P now does not include the set of p since corresponding tuples of R2 are already received in $R2'_p$. In probe step we use $R2'_p$ together with R2' and the last step is materialization of T.

As a comment on the algorithm; we should say that the size of p has impact on the accuracy of the estimation; if we increase the size, the estimation becomes more accurate. However, large volume of p may result in more overhead with this initial investigation mechanism. This sampling should be done in a way such that the page is a good representative of R1 and $R2'_p$. Estimation accuracy of the selectivity factor which is the ratio of R1 tuples corresponding to R2 tuples depends on the accuracy of the decision criteria embedded in sample (P) step.

```
SMDJoin (R1, R2, T: Relation; V, X: Set of Attributes)
{
    If (not local (R1)) then receive (R1);
    Build (R1, HT1);
    P ← Project (R1, V);                    // compute statistics on R1 and P
    p ← sample(P);                          // select some tuples
P1
    If (not local (R2))  then
        {
            send p  to S2;
            //  DAccess runs on the remote site
            receive R2'p  from S2;
        }
        else
            DAccess(R2, p, R2'p);
P2                                          // revise statistics on R2'p

    Site ← Decision (StatInfo,
                     DataAvailabilitiy,
                     SystemState);
    If (not local (Site) then migrate on Site;

    If (not local (R2))  then
        {
            send (P-p)  to S2;
            //  DAccess runs on the remote site
            receive R2' from S2;
        }
        else
            DAccess(R2, (P-p), R2');

    Probe (HT1, (R2'U R2'p), T);
    If (not local (T) then send (T) else materialize (T);
```

Figure 3.6: Algorithm of a Sampling Mobile Dependent Join (SMDJoin)

In this section we have demonstrated the use of mobile agents in query execution with operators specifically designed for restricted sources. These operators are capable to self-adapt dynamically to the variations between the compile-time estimation and run-time computation of the parameters of data. Mobile query execution model presented in chapter 2

of the thesis becomes capable of handling the source restrictions met in large scale data integration environment with these new operators.

In next section we will propose a solution to another weakness related with using mobile query execution model: this solution aims to bridge the mobile query execution model and the initial placement of the optimizer. We will discuss the limitations of the placement strategies of current optimizers with mobile relational operators and propose an approach for the placement of them. As already indicated, they are sensitive to the site they are placed: initial site should not prevent their adaptation (migration) during execution.

3. Initial Placement of Mobile Relational Operators for Large Scale Distributed Query Optimization

In this section we will start by focusing on the behavior of mobile relational operators and the problems of single point placement strategies with respect to placement of these operators on the sites and give the problem position. After then, we will introduce the components required for initial placement of mobile relational operators: migration space, robust site and robust placement functions.

3.1 Mobile Relational Operators and Current Placement Methods

We will start our discussion by abstracting the characteristics of mobile relational operators (MROs) and continue presenting the problem related with current placement approaches with MROs.

3.1.1 Characteristics of Mobile Relational Operators

We want to discuss the algorithms of mobile join (section 2.5), MDJoin and SMDJoin (section 3.2). Mobile join is capable to handle positive query where the data sources do not have limited query capabilities. It works in coordination with abstracted Scan operator on the sites of the data sources. It receives R1 from the first data source and constructs the hash table. With better estimation it makes its decision to migrate on a new site or not.

MDJoin and SMDJoin are mobile join operators as well but they are designed for negative query where sources impose different limited query capabilities. Their restrictions are abstracted by means of binding patterns (section 2.3.1). They work in coordination with abstracted DAccess operator on the site of the source imposing access restriction and abstracted Scan operator on the site of the data source imposing no access restriction. In a similar manner of the mobile join, MDJoin and SMDJoin receive R1 from the first data source and construct the hash table. MDJoin projects the attributes corresponding to bound attributes of the restricted source. With better estimation it makes its decision about migration. On the other hand, SMDJoin after projection of the bound attributes makes a sample on this projected table. With this sample table it receives the corresponding tuples of R2 from DAccess operator of the restricted source. It now has better estimation and makes its decision to migrate or not. Both of these operators: MDJoin and SMDJoin receive data from the restricted source by DAccess operator.

After migration step, all the three operators behave in a similar manner; they receive R2 (by Scan or DAccess) and probe the hash table with R2 or R2'. If they are on final site (site expecting the result of the join) they materialize or send the temporary relation to final site.

The main difference between the mobile join of positive query and MDJoin and SMDJoin for negative query lies in their: (1) coordination with Scan operator with both data sources or coordination with DAccess operator on the data source posing access restriction and (2) before migration projecting the bound attributes required for the restricted source or not. These differences in their algorithms do not change their sensitivities to the initial site they are mapped. They are in need of a site which is defined taking into the account the estimation errors on the size of R1 and a site which would not obstruct their migration.

3.1.2 Limitations of Single Point Placement Methods

Present single-point placement methods [FRA 98, LIU 98, ROD 00 and ZHO 05] try to minimize the communication cost by placing the relational operators on the sites minimizing the data transfer by taking into account single-point estimates instead of an

estimation interval. Let us demonstrate the problem with a simple query: T ← Join(R1, R2), where R1 and R2 are base or temporary relations residing on the sites S_1 and S_2 and the site of T is S_1. Suppose the optimizer decides to place initially the mobile join operator, MJO on S_2 with the estimation |R2| > |R1| + |T|. If the compile-time estimate of the optimizer is correct, the response time of the query on S_2 is minimized. However, if the compile-time estimate is not correct and for example |R1| + |T| > |R2|, in this situation, the MJO loses possibility of migration because its migration triggers the transfer of its hash table and R2. In the case of restricted R2; mobile join makes its decision depending on the estimation |R2| > |R1| + |T| + |P| (P is the table projected from R1 from corresponding the values of bound attributes).

Our simple test shows (Figure 3.7) that with P-S2 (MJO is placed initially on site S_2), the increase in the response time of the MJO might be more than 220% compared to the best placement which is on site S_1 (P-S1). On the other hand, if the optimizer places the MJO on site S_1 and if the compile-time estimate is correct, the MJO achieves response time which is more than P-S2 with the overhead of migration (the cost of serialization, transfer and de-serialization of the MJO) which is shown around 6% of increase in response time [OZA 05a and OZA 05b]. However, if the compile-time estimate is not correct, MJO can decide to stay on S_1 and has better performance. These simple example shows, with a risk of loosing 6% in response time, it is possible to gain more than 220% in cases of estimation errors.

3.2 Robust Placement

In this subsection we will first introduce the components of robust placement method and then explain a robust placement algorithm. Robust placement method is based on two main components: i) migration space: set of sites to which mobile relational operator is allowed to migrate and ii) robust site: site which allows acceptable performance within an estimation interval instead of single points.

3.2.1 Migration Space

It is not reasonable to allow a MRO to migrate to all existing sites over large scale environment. The time to compute the better site among all the sites present on the network

would be prohibitive. So, the optimizer should define a subset of sites on which the MRO can migrate during its execution: migration space. This one should be defined taking into account the migration spaces of the neighbor operators.

Migration space of a MRO is defined as the set of execution sites that the MRO can migrate during its execution. Migration space should include the sites of the producer and consumer operators [FRA 98 and LO 93]. Consider the previous example and assume that R1 and R2 are on the sites S_1 and S_2 respectively and the result of the query T is expected on site S_1. Hence, the producer operators like Scan(R1) and Scan(R2) should be on the sites S_1 and S_2 respectively. The site of T is site S_1. Then, the migration space MS of MRO is quoted $MS_{MRO} = \{S_1, S_2\}$.

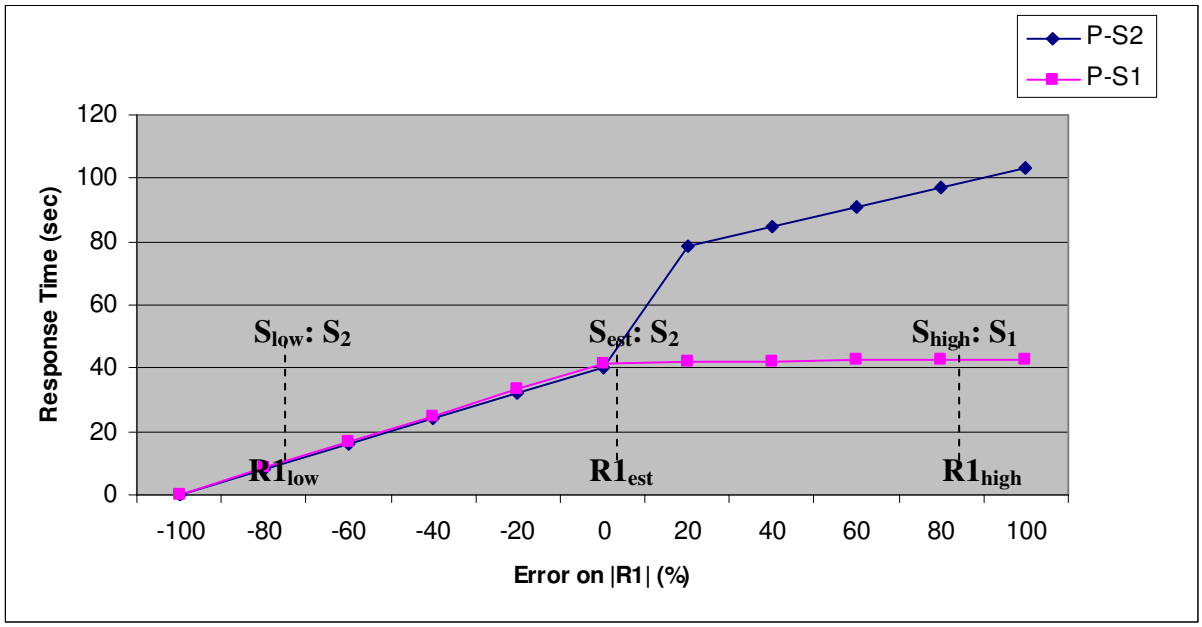

Figure 3.7: Response times of PS-1 and PS-2 with respect to error on |R1|

3.2.2 Robust Site

Figure 3.7 shows the response times of two execution plans of the same MJO with respect to the error on |R1|. P-S1 is the plan in which MJO starts its execution on site S_1. P-S2 is the plan in which MJO starts its execution on site S_2. In both plans MJO continues its execution on site S_1 or S_2. Migration space of the MJO is defined as $MS_{MJO} = \{S_1, S_2\}$. We

assume to have three points defined by the optimizer corresponding to the estimated size of the relation R1: $R1_{low}$, $R1_{est}$ and $R1_{high}$ represent respectively the cases of smallest, average and highest estimations of the optimizer on |R1|. Inside the interval [$R1_{low}$, $R1_{est}$] both plans P-S1 and P-S2 achieve close response times to each other. However, P-S1 incurs a slight extra cost due to migration overhead-threshold which is found around 6% increase in response time [OZA 05a] in our case. For the interval [$R1_{est}$, $R1_{high}$], P-S1 is not affected by the increased size of R1. On the other hand, if we look at the cost of P-S2, we remark that the response time of P-S2 increases dramatically since its migration causes transfer of hash table and R2. As a result, for the interval [$R1_{low}$, $R1_{est}$], S_2 is the site giving minimum response time ($S_{low} = S_2$), for $R1_{est}$, S_2 is the site giving minimum response time ($S_{est} = S_2$) and for [$R1_{est}$, $R1_{high}$], S_1 gives the minimum response time ($S_{high} = S_1$).

This simple example demonstrates that the execution plans are quite related with the initial placement and not all placements provide acceptable performance over an estimation interval. In order to find a site giving acceptable performance over an estimation interval we should check the response time of the MRO for all the points of the estimation interval. In our case, response times of P-S1 for $R1_{low}$ and $R1_{est}$ is closed to minimum with the threshold, and for $R1_{high}$, PS-1 is the plan with a minimum response time. Hence, the site S_1 provides acceptable performance over an estimation interval. We say that it is a robust site for MRO.

In order to formalize the definition of a robust site, we associate to each MRO the following annotation: (i) three estimation points $R1_{low}$, $R1_{est}$ and $R1_{high}$, (ii) a migration space defined as $MS_{MRO} = \{S_1, S_2, \ldots, S_n\}$, (iii) S_{low}, S_{est} and S_{high} corresponding to the sites minimizing the response time corresponding the values of $R1_{low}$, $R1_{est}$ and $R1_{high}$ respectively, (iv) a response time set = {RT (S_i, $R1_k$)} meaning that the response time achieved by initially placing MRO on S_i with the estimated size of $R1_k$, for $k \in$ {low, est, high} and $S_i \in MS_{MRO}$ and (v) mobility overhead constant of the optimizer defined as threshold.

A site S_i is said to robust iff:

$\forall k \in$ {low, est, high} and $\forall RT(S_i, R1_k) <= RT(S_k, R1_k) *$ threshold.

A site is robust if and only if, for all estimated points of R1, the response time of MRO initially placed on it, is equal to or close to the response time of the site giving minimum response time for this R1 estimation with the threshold. In other words, MRO should have a response time that is close to the response time of the site giving minimum response time for all estimated points in the interval. If we go back to the situation of S_1 and S_2 of Figure 2 we can say that S_1 is robust site since its response time for $R1_{low}$, $R1_{est}$ and $R1_{high}$ is equal or close to the minimum response times of S_{low}, S_{est} and S_{high} with the threshold.

```
CheckRobustness(site, Node, threshold): Boolean {                    // threshold is the mobility overhead
Return (RT(site, Node.R1low) ≤ RT(Node.sitelow, Node.R1low) * threshold and
            RT(site, Node.R1est) ≤ RT(Node.siteest, Node.R1est) * threshold and
            RT(site, Node.R1high) ≤ RT(Node.sitehigh, Node.R1high) * threshold)}

RobustSite(Node, threshold) {
    CheckRobustness(site, Node, threshold): Boolean;
    // SiteMinRT function returns the site giving the minimum response time with respect to given estimation point

    Node.sitelow  ← SiteMinRT(Node, Node.migrationSpace, Node.R1low);
    Node.siteest  ← SiteMinRT(Node, Node.migrationSpace, Node.R1est);
    Node.sitehigh ← SiteMinRT(Node, Node.migrationSpace, Node.R1high);

    if  (CheckRobustness(Node.sitelow, Node, threshold)) then return Node.sitelow        // sitelow : robust site
        else if (CheckRobustness(Node.sitehigh, Node, threshold))  then return Node.sitehigh   // sitehigh : robust site
                else return Node.siteest;
}
```

Figure 3.8: Algorithm of the Robust Site

The algorithm of the robust site function given in Figure 3.8, takes a Node (MRO) as an input and returns a robust site. The robustness of each site is controlled with the CheckRobustness function which takes a site, a MRO and a threshold as input and returns true only if the response times of the MRO for $R1_{low}$, $R1_{est}$ and $R1_{high}$ on the site are equal or close to the response times of S_{low}, S_{est} and S_{high} with the threshold respectively. If S_{low} and S_{high} do not satisfy the requirements of robustness, then the chosen site is S_{est}.

3.2.3 Robust Placement of Mobile Relational Operators of a Query

The operators of a query execution plan are mobile binary operators, scans (as explained in section 2.5) and filter operators (e.g. select and project). Binary operators are mobile joins: mobile joins as explained in section 2.5 (for positive query) and MDJoin's or SMDJoin's as explained in section 3.2 for negative query.

```
RobustPlacement(Node, resultSite, threshold){

    if (isScan?(Node)) then        {Node.migrationSpace ← Node.R1.Replicates;
                                    Node.site ← SiteMinRT(Node, Node.migrationSpace, Node.R1est);}

    if (isFilter?(Node)) then {RobustPlacement(LeftChild(Node), resultSite, threshold);
                               Node.migrationSpace ← LeftChild(Node). migrationSpace;
                               Node.site ← LeftChild(Node).site;}

    if (isBinary?(Node)) then      {RobustPlacement(RightChild(Node), resultSite, threshold);
                                    RobustPlacement(LeftChild(Node), resultSite, threshold);
                                    Node.migrationSpace ← RightChild(Node).migrationSpace ∪
                                                          Leftchild(Node).migrationSpace ∪ resultSite;
                                   Node.site ← RobustSite(Node, threshold);}
}
```

Figure 3.9: Algorithm of the Robust Placement

The robust placement algorithm, Figure 3.9, places a MRO together with its children. It has as input a MRO, the site where the query result is expected and a threshold. It is a recursive algorithm which places first the right and left child of MRO.

If the Node is a Scan operator, the placement space is defined as the sites where its operand is replicated. It is mapped on a site of the placement space providing the required data with a minimum response time. For a filter operator, its migration space is defined to be the migration space of the direct child operator, and it is mapped on the placement site of the direct child operator. When the operator is binary operator, the placement of the right child and left child is done first. Then the migration space of the binary operator is defined and

finally the robust site is decided by the RobustSite function among the sites present in its migration space.

4. Conclusion

In this chapter we have presented two studies in the context of mobile query execution model with decentralized adaption style which makes its applicability over large scale network possible. The characteristics of the environment like high network latency and low bandwidth lead way to (1) decentralized methods in order to reduce the amount of message passing towards a single control authority and (2) to the adaptation methods paying attention to the amount of the volume of data transferred. *Perspective* of the studies presented in this chapter is to extend the mobile query execution model (1) as to capture the requirements of the data sources with access restrictions which are met in data integration environment and (2) as to make robust initial placement of mobile relational operators which are sensitive to their initial sites.

First study in the direction of *perspective (1)* proposes the use of mobile agents in query execution operators specifically designed for restricted sources. These benefit from autonomous and reactive features of mobile agents in a way to self-adapt dynamically to the variations between the compile-time estimation and run-time computation parameters on data. We introduced two mobile join operators: mobile dependent join (MDJoin) and sampling mobile dependent join (SMDJoin) which are designed for restricted sources. These two mobile relational operators proposed are capable to deal with restricted sources and react to the variations between the compile-time estimations and run-time parameters. MDJoin makes its adaptation decision (better site to continue its execution) relying on the true statistics of R1 and better estimation of T whereas SMDJoin includes a sampling step. With the help of this additional step its adaptation decision is based on true statistics of R1 and better statistics of T and R2'. Two new operators designed for restricted sources: MDJoin and SMDJoin extends mobile execution model to capture the requirements of the restricted sources in data integration environment. Another required extension to mobile query execution model comes with the second study for the perspective (2) is about the initial placement of mobile relational operators which are sensitive to their initial places.

Robust placement method, to map in compile time, the mobile query operators on execution sites over large scale network where the data sources are unpredictable is presented by the second study in this chapter. We devised i) an approach to determine the migration space of a mobile relational operator which includes the sites on which the mobile relational operator is allowed to migrate during its execution, ii) a function to determine the robust site which will allows acceptable response time equal or close to the minimum response time for the estimation points in the interval and finally iii) placement algorithm which places the operators of the query execution plan tree in a bottom up way. The robust placement approach with these components makes the placement of mobile binary operators on execution sites based on an estimation interval. A site which allows acceptable performance over an estimation interval is chosen instead of a site giving the best performance on a single estimation point.

In next chapter we will first explain the simulation environment. Then we will present the performance evaluation studies of the methods presented in this chapter: mobile join operators designed for restricted sources and initial placement of mobile relational operators.

Chapter IV Performance Evaluation

1. Introduction

Objective of this chapter is to evaluate the effectiveness of the methods presented in chapter 3. We will start by explaining our simulation model. In section 2, we show the results of the performance evaluation on the mobile execution model for restricted sources. In this perspective, the performances of the operator of standard strategy: dependent join (DJoin) and the operators of decentralized adaptive strategy: mobile dependent join (MDJoin) sampling mobile dependent join (SMDJoin) are analyzed with respect to the estimation error on the size of data and run time parameters like CPU frequency and network bandwidth. In section 3, we compare robust placement strategy with single point placement strategy with mobile relational operators in two main scenarios: single join and multi-join. We study the behavior of both methods when there is error on the estimations related with the size of the relations and the selectivity factor of the joins. In section 4, we make our conclusion remarks on the findings of two performance evaluation work.

Simulation Model: Simulator is composed of two parts: simulated method and architecture. Disk, CPU, and memory characteristics describe each site of the target architecture. We assume that characteristics of all the sites and communication links are the same. This time, site failures and data unavailability are not considered even if the risk is high in large scale distributed systems. Finally, it is assumed that buffer size of the relational operator is unlimited. The main costs associated with the basic operations and with the parameters are given in Table 4.1, according to [MOR 02] and to [HAG 00] for the approximation of the duration of the migration of the relational operator. As the size of the hash table is varying, the simulator computes the duration of the hash table serialization from the simulation parameters.

Disk parameters	Disk page size	4 KB
	Average time to read a 4-KB page	47.5 μs
	Average time to write a 4-KB page	195 μs
CPU parameters	Pentium III processor	550 MHz
	Memory size	512 MB
Network parameters	Maximum bucket size	4 KB
	Time to send a 4-KB page	50 ms
	Latency	20 ms
Miscellaneous	Average number of tuples in page	32
	Agent size	2806 B
	Duration of agent migration[2]	150 ms

Table 4.1: Simulation parameters

2. Performance Evaluation of Mobile Relational Operators for Restricted Sources

In this section, we study the performance evaluation of the operator of standard strategy, DJoin and the operators of dynamic decentralized strategy, MDJoin and SMDJoin. The focus of the evaluation is on their performance with respect to the following parameters: response time in the presence of the variations between the compile-time estimations and the run-time computations of the sizes of the relations, speedup[3] realized in comparison with the network bandwidth and CPU frequency. Special emphasis is given to the variation between the compile-time estimation of the optimizer and the run-time computation of the parameters since errors in operator level reveals out propagation effect on the error in the query level [IOA 91]. Cost of code transfer is not taken into account (codes are supposed to be resident on mediator sites hosting mobile agents; additionally, it is of little significance compared to the transfer cost of tuples). A convenient mechanism for data serialization without specific overhead is assumed to exist as well.

To have quantitative comparison, R1 is considered to be a temporary relation (i.e. stemming from two sub-queries) and R2 is considered to be restricted source which can only be accessed by DAccess operator. Their sizes are estimated by the optimizer as 10.000 tuples

[2] The duration of agent migration includes serialization of data, transfer of the serialized data and the execution state and de-serialization of data

[3] The speedup: response time of a DJoin / response time of a mobile join operator

for R1 (on site S_1) and 20.000 tuples (on site S_2). The estimated selectivity factor is 1.5/20.000 [SHE 93]. Hence estimated size of the result relation is 15.000 tuples. The size of p for SMDJoin operator is 512 tuples, it is kept fixed in all cases since seeing the impact of the size of p is beyond the objective of this evaluation.

2.1 Impact of the Variation of the Sizes of Data

The behaviors of DJoin, MDJoin and SMDJoin are analyzed in the cases of decreasing and increasing size of the data of both sides of the join operation, in order to see the behaviors of the operators when the optimizer under or over-estimates at compile-time.

- **Variation of the Size of R1**

In the experiment of Figure 4.1, the response time depending on the variation between the compile-time estimation and the run-time computation of the size of R1 is analyzed by decreasing the number of R1 tuples; as a consequence P is decreased. The result relation is stored on the site S_1, the optimizer plans to build the hash table on site S_1 and to transfer R2' to site S_1. DJoin, MDJoin and SMDJoin are located on site S_1 (for build phase). Up to -30% of variation DJoin performs slightly better than MDJoin and SMDJoin since mobile relational operators have some overhead. After -30% of variation between compile-time estimation and run-time computation of the size of R1, MDJoin and SMDJoin are more efficient. From this threshold, the global volume of data to be transferred is less if the join migrates. We see that MDJoin and SMDJoin behave in a similar manner for the cases when the variation of the size of R1 is greater than 30%.

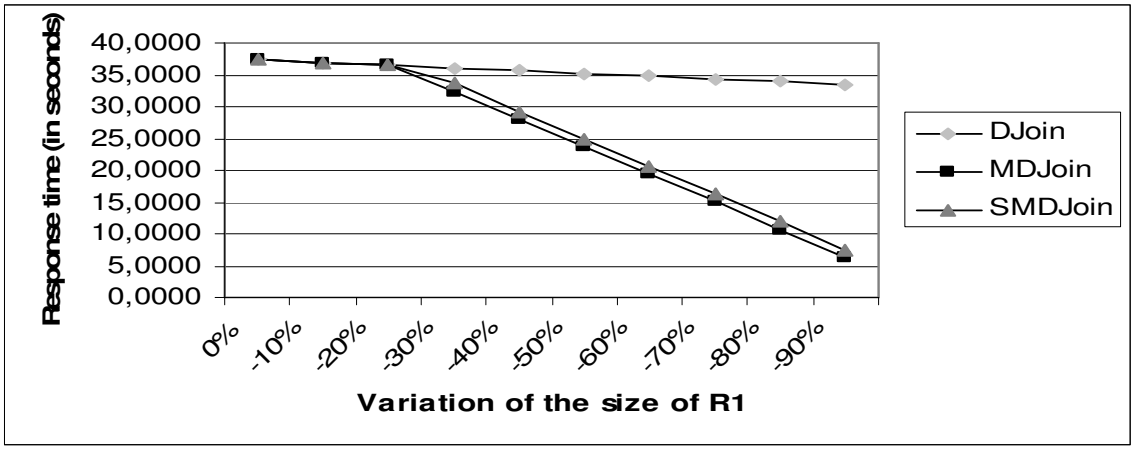

Figure 4.1: Response time with decreasing size of R1 (becoming less than the estimation)

In the second test, the impact of the variation between the compile-time estimation and the run-time computation of the size of R1 is evaluated by increasing its size; the size of P is increased as well. The optimizer plans to transfer R1 to the site S_2, and the result relation must be stored on another site. In this case the transfer cost of the result relation is not estimated. As seen in Figure 4.2, between 10.000 and 20.000 tuples of R1, the response time of MDJoin and SMDJoin are slightly higher than DJoin. This is due to the agent migration time from site S_1 to S_2. While DJoin is located on site S_2 at all cases, mobile relational operators are located on site S_1 after the size of R1 exceeds the size of R2' (which is known to be 20.000 tuples). After 20.000 tuples of R1, MDJoin and SMDJoin decide to stay on site S_1 and transfer R2', while DJoin continues to transfer R1 to S_2. Hence, the MDJoin and SMDJoin reduce the response time by 20% to 40% by minimizing the transferred data volume. Again the behaviors of MDJoin and SMDJoin are quite similar for this case as well.

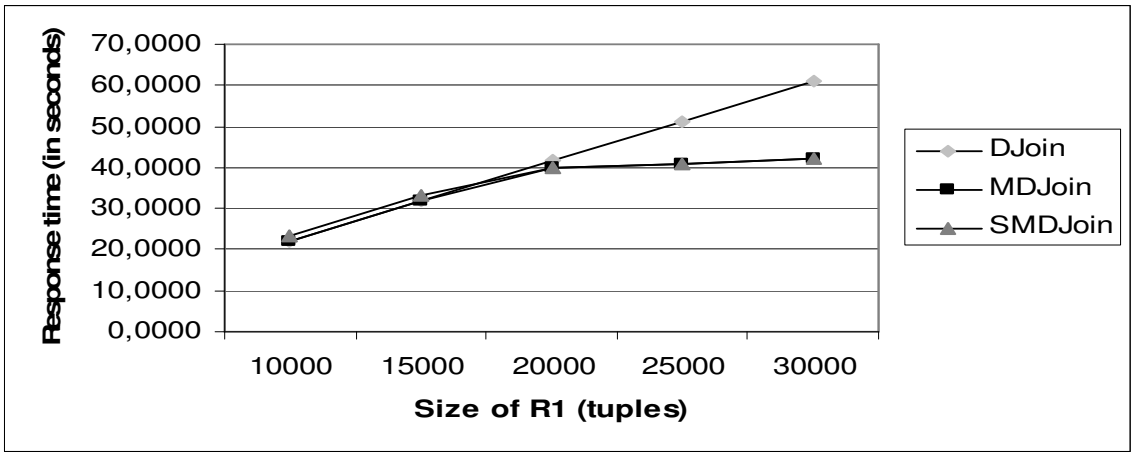

Figure 4.2: Response time with increasing size of R1 (exceeding the estimation)

- **Variation of the Size of R1 and R2'**

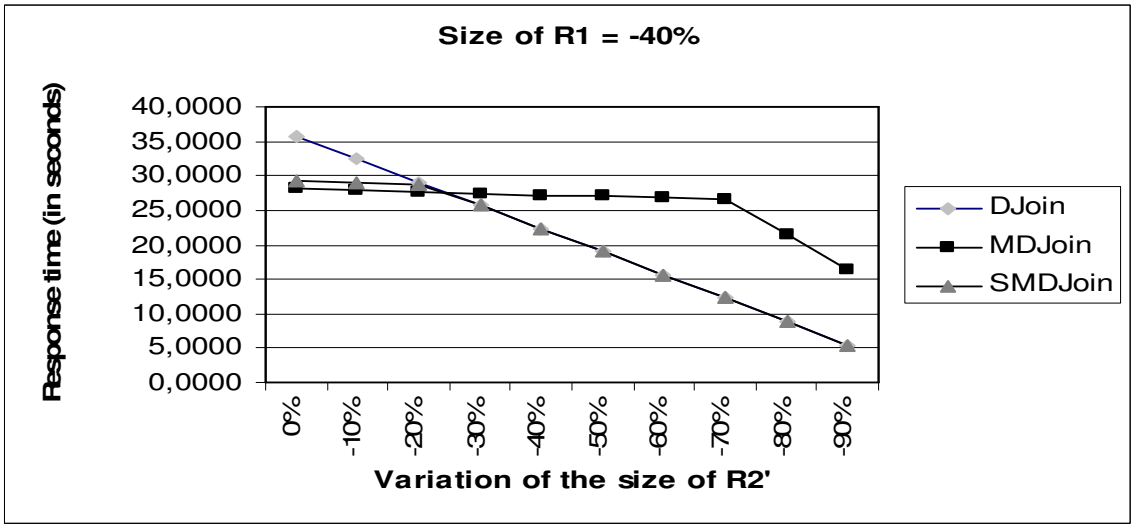

Figure 4.3: Response time with decreasing size of R2' (becoming less than the estimation)

In Figure 4.3 we see the size of R2' becoming less than expected for the case of the variation of the size of R1 is -40%. Up to -30% of variation of the size of R2' both MDJoin and SMDJoin perform slightly worse than the DJoin since agent migration brings an overhead. After 30% of variation, it is not a good decision to move so MDJoin (not seeing the variation between the compile-time estimation and the computed size of R2') moves and

performs worse than DJoin and SMDJoin. DJoin is located on the correct site by chance. SMDJoin with the help of its sampling step chooses the right site.

The results observed from Figure 4.4 are similar to the above case. Since the variation of the size of R1 now is bigger than the above case, -60%, to choose the right site is even more important. Superiority of SMDJoin becomes more evident since it chooses the right location for all the variations between the compile-time estimation and run-time computations of the sizes of R1 and R2'.

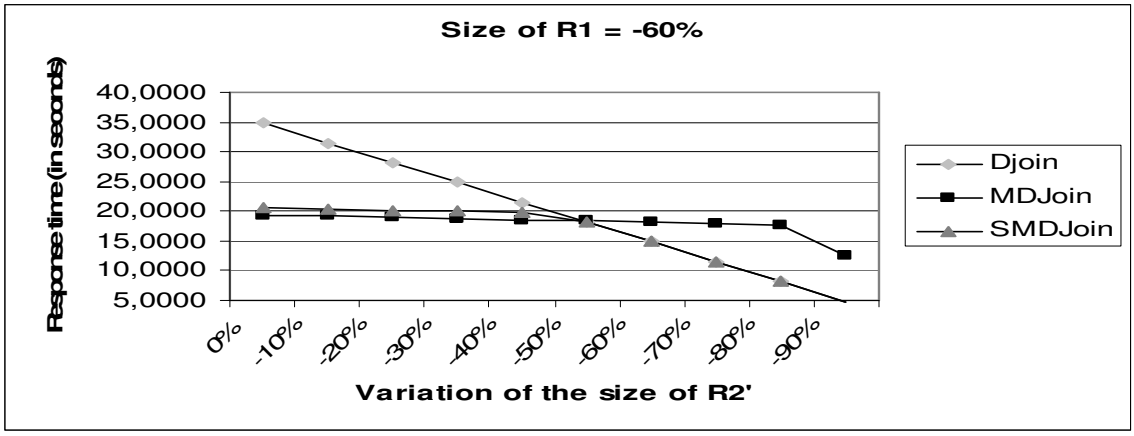

Figure 4.4: Response time with decreasing size of R2' (becoming less than the estimation)

Figure 4.5 demonstrates the case of R2' being more than expected. We see that DJoin and SMDJoin behave similarly after the size of R2' exceeds the estimated size since DJoin is already on site S_2 and SMDJoin sees the variation between the compile-time estimation and run-time computation of the size of R2' and stays on site S_2. The performance of MDJoin gets worse since it is blind to the variation of size of R2' and decides to move R2'. When R2' is bigger than expected it is not a good decision to move.

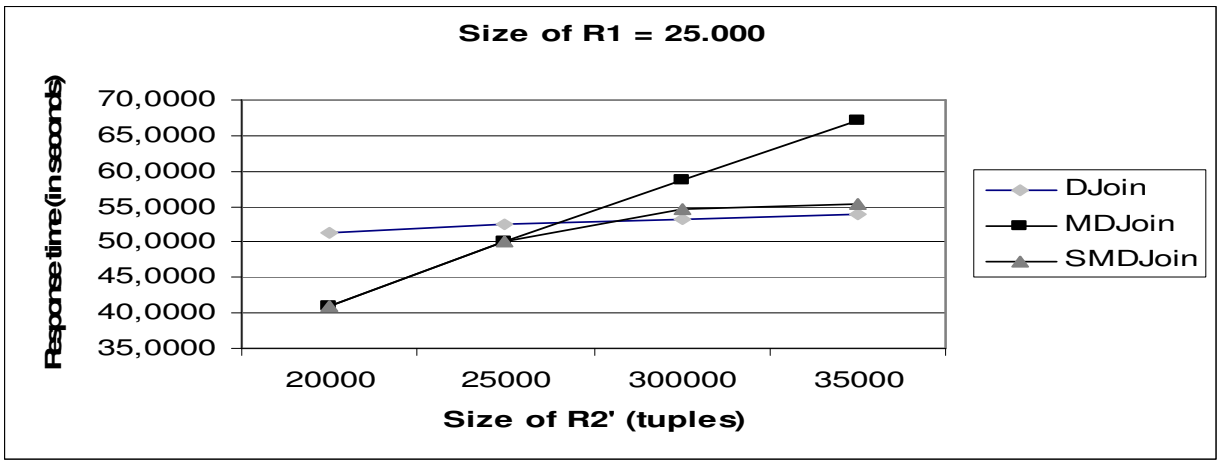

Figure 4.5: Response time with increasing size of R2' (exceeding the estimation)

2.2 Impact of the CPU Frequency and the Communication Bandwidth

In this section we analyze the impact of the CPU frequency and the communication bandwidth on speedup realized by the two mobile relational operators; MDJoin and SMDJoin. Figure 4.6 shows the speedup of MDJoin for several levels of variations between the compile-time estimation and the run-time computation of the size of the R1. We see that up to 1 GHz there is a speedup. After 1 GHz, speedup is limited by network bandwidth that constitutes a bottleneck.

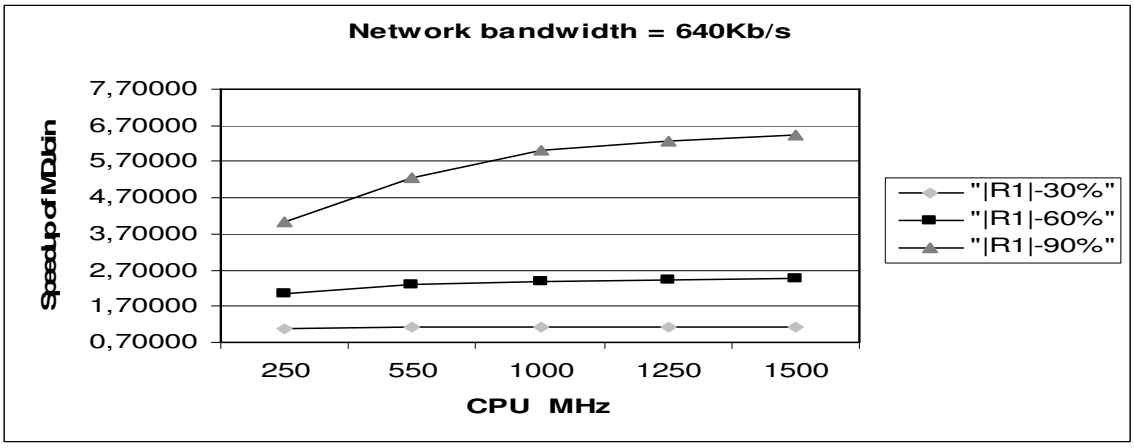

Figure 4.6: Speedup of MDJoin with increasing CPU frequency for different variations of the size of R1

Figure 4.7 shows the speedup of SMDJoin for several levels of variations between the compile-time estimation and the run-time computation of the size of R1. Again, we see that up to 1 GHz there is a speedup in relation to CPU frequency. Figure 4.8 is the comparison of the speedups realized by MDJoin and SMDJoin depending on the CPU frequency when the variation between the estimation of the optimizer and the size of R1 = -60%. We see that SMDJoin achieves slightly less speedup due to overhead caused by its sampling step.

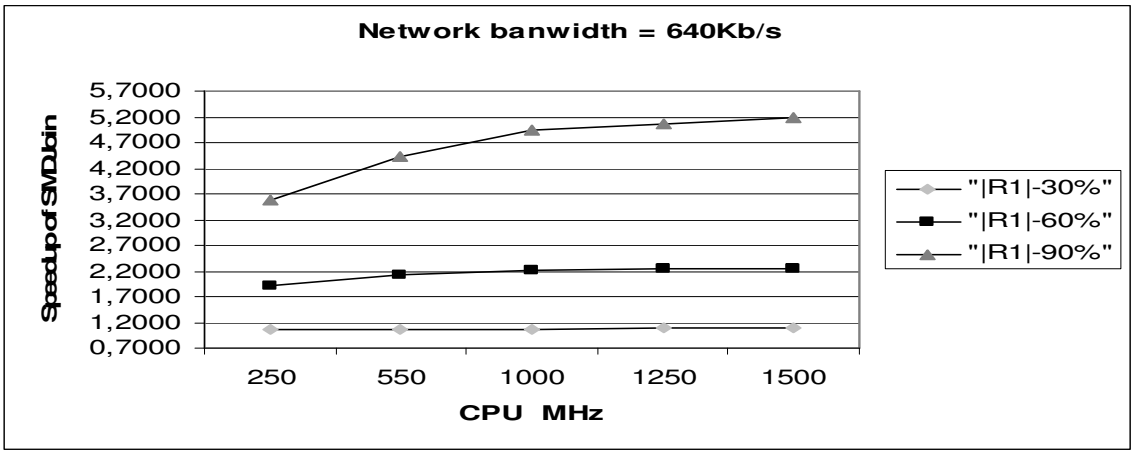

Figure 4.7: Speedup of SMDJoin with increasing CPU frequency for different variations on the size of R1

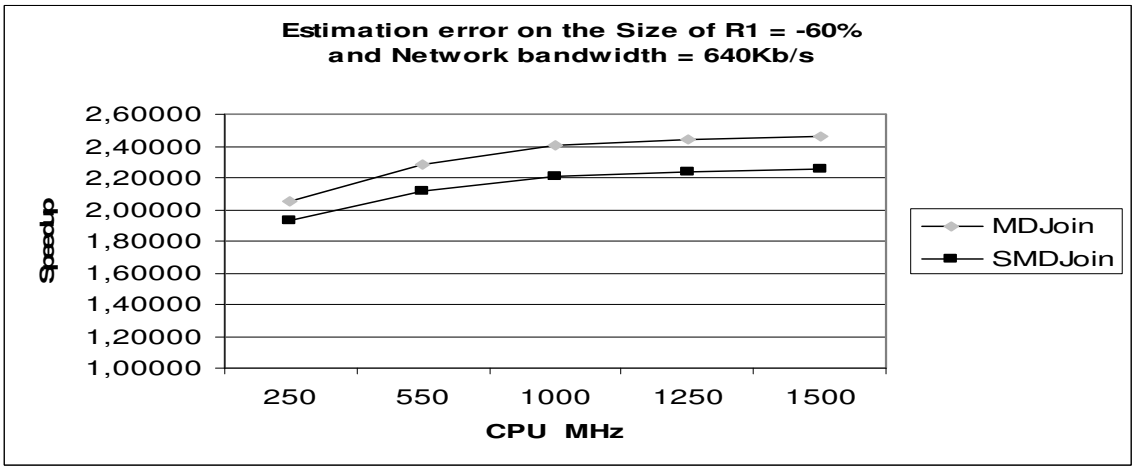

Figure 4.8: Speedup of MDJoin and SMDJoin with increasing CPU frequency for variations on the size of R1 = -60%

Figure 4.9 shows speedup realized by MDJoin operator for several levels of variation between the compile-time estimation and run-time computation of the size of R1. It can be seen that speedup decreases after 640 Kb/sec to 1.28 Mbs/sec of bandwidth confirming the advantage of mobility for bandwidth values less than 640 Kb/sec which is quite likely to happen in large scale environment.

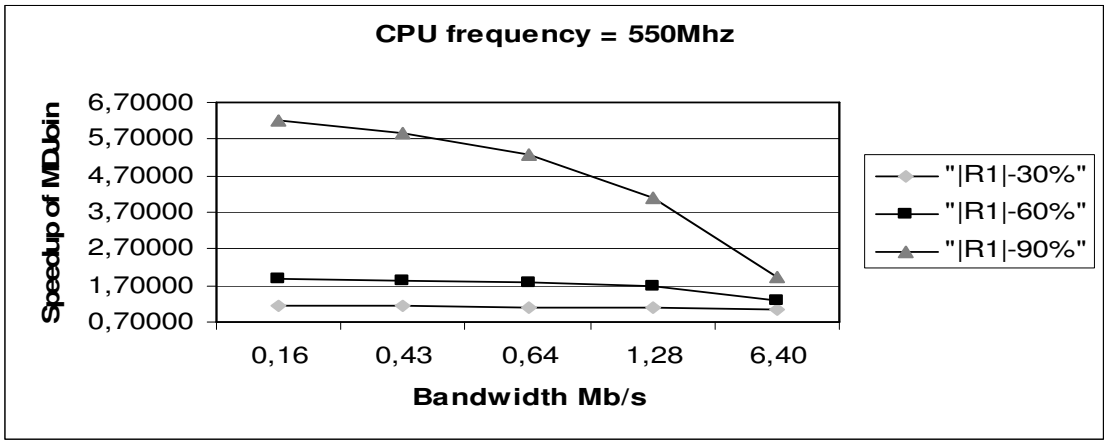

Figure 4.9: Speedup of MDJoin with increasing communication bandwidth for different variation of the size of R1

Figure 4.10 shows speedups realized by SMDJoin operator for several levels of variation between the estimated and computed size of R1. Results are quite similar to those of the above case.

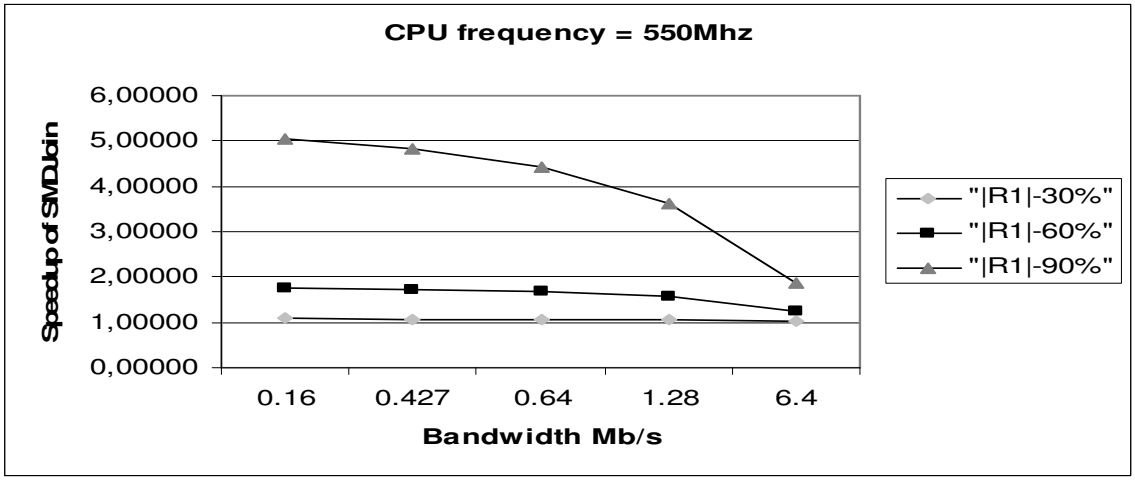

Figure 4.10: Speedup of SMDJoin with increasing communication bandwidth for different variation of the size of R1

Figure 4.11 is the comparison of the speedups of MDJoin and SMDJoin operators depending on the communication bandwidth when the variation of the size of R1 = -30%. We see that SMDJoin brings slightly less speedup due to its sampling step.

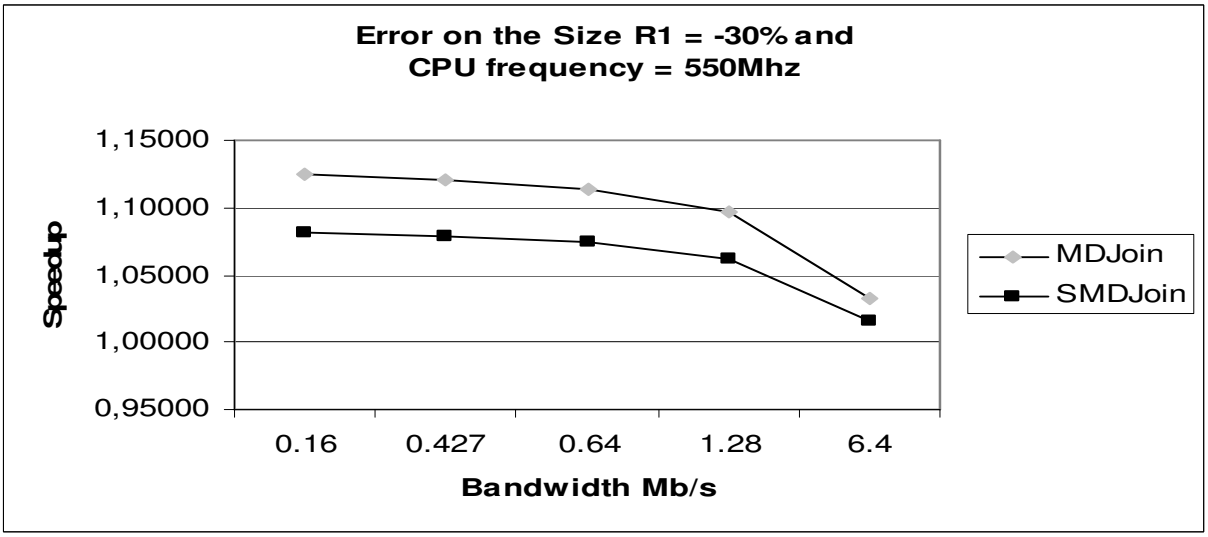

Figure 4.11: Speedup of MDJoin and SMDJoin with increasing communication bandwidth for variation of the size of R1 = -30%

2.3 Discussion on the Performance Evaluation

The results demonstrated that if the size of the data from the first source is more than 30% different than the estimation made by the optimizer, our mobile dependent operators have better performance compared to the operator of the standard strategy. Against the variations of the size of the data coming from the second source SMDJoin is superior to MDJoin, since due to its sampling step it sees the variation of the size of the data of the second source. The overhead of this sampling step for SMDJoin is less than 10 % of increase in response time compared to MDJoin for the cases where they both make the same decision for the location of the probe step. But if the decision for the site to continue the execution is wrong (for MDJoin) response time of MDJoin is higher by up to 4 times depending on the variation between the estimated and computed sizes of R1 and R2' than the one of SMDJoin.

For measuring the effect of the CPU frequency on the response times of the standard and mobile strategy, we executed our operators with different levels of the variation between the compile-time estimation and run-time computation of the size of the data of the first source. The results demonstrated that there is an increasing speedup generated by the mobile relational operators up to 1 GHz of CPU frequency. After 1 GHz speedup is limited by network bandwidth that constitutes a bottleneck.

Experiments on the impact of the communication bandwidth were done with similar approach as the ones of CPU frequency and the results show that there is a speedup succeeded by mobile relational operators up to 640 Kb/sec of communication bandwidth. The advantage of mobility for the cases where the communication bandwidth is probably less than 640 Kb/sec is evident.

Although the response times of our performance evaluation are not the same, the shapes of the most of the curves related with MDJoin are similar to performance results of mobile join operator of [ARC 04]. Due to the inadequacy of the scale granularity it is not possible to demonstrate the impact of computing and/or transmitting set of P and retrieving R2' through DAccess operator. However, it is not wrong to say that the behavior of mobile join of and MDJoin are similar. The behavior of SMDJoin is different than the mobile join

and MDJoin when there is a variation between compile-time estimated and run-time computed data parameters of R2'.

3. Performance Evaluation of Initial Placement of Mobile Relational Operators

In this section, we compared single-point and robust placement methods mapping single join and multi-join. We study the behavior of both methods when there is error on the estimations related to the size of relations. Performance study is based on a simulation model explained in sub-section 1.1 of this chapter.

3.1 Single join

Let us consider two relations R1 and R2 residing respectively on sites S_1 and S_2 with number of tuples estimated as 10.000 and 30.000. Also, let us assume a simple hash join J: T = Join(R1, R2). The result is expected on the site S_1. Migration space of the join MSJ= {S_1, S_2}. The estimated selectivity factor [SHE 93] of join J is 1.5/max(||R1||, ||R2||) where ||R1|| and ||R2|| indicate the number of tuples. The expected threshold of the optimizer capturing mobility overhead is 1.06 increases in response time of the join operation. Our performance evaluation measures the influence of the errors on the |R1| (i.e. size of R1) on response time with single-point and robust placement methods. The size of R1 is varied from -80%, (80% smaller than the size estimated at compile-time) to 160%, (the size is 160% bigger than the size estimated at compile-time).

Figure 4.12 demonstrates the impact of the error on |R1| on response time of the join operation. Single-point placement method places the MJO on site S_2 since estimated |R2| > |R1| + |T| whereas robust placement method places the MJO on site S_1 since S_1 is the robust site for $R1_{low}$, $R1_{est}$ and $R1_{high}$ which are estimated by the optimizer as 256 KB, 1 280 KB and 3 328 KB respectively. We observe clearly that the response time of robust placement is higher than the response time of single-point placement for the range [-80%, 20%]. In this range, the difference between two response times is between 2,9% and 3,7% which is less

than the threshold. On the other hand, for the range of [20%, 160%], the speed up achieved by robust placement method reaches up to 300 % when compared with the response time of single-point placement method. So with the risk of loosing 3.7% in response time, it is possible to increase the performance by 300% approximately.

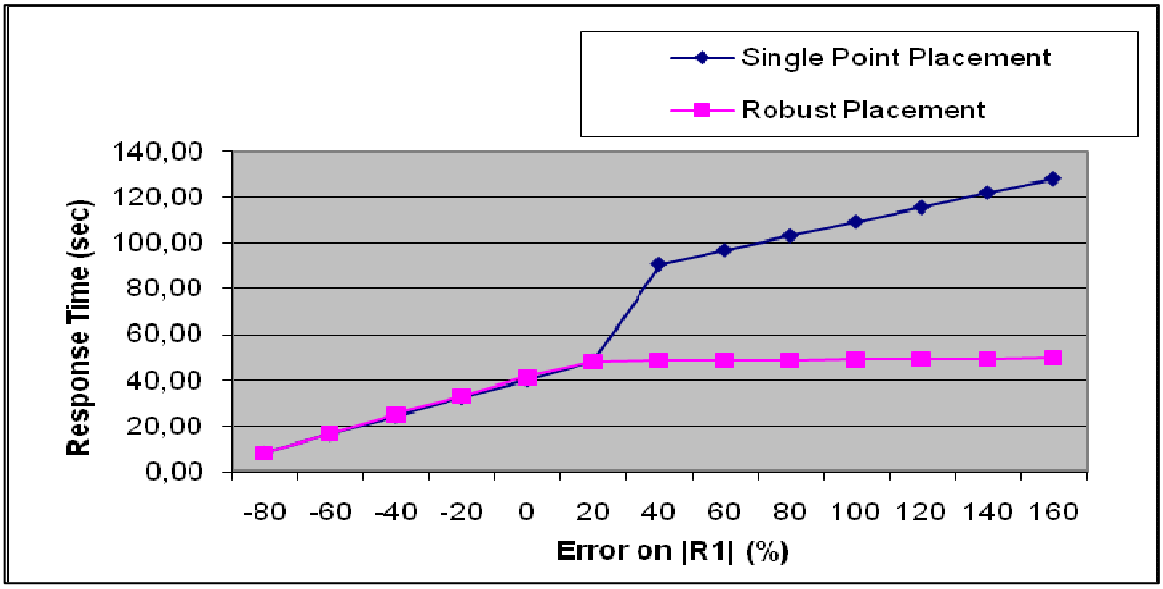

Figure 4.12: Response time with respect to error on |R1|

Figure 4.13 demonstrates the impact of the error on the estimated constant s of the selectivity factor calculation. This variation affects the size of the result relation T; similar of the approach presented above for $R1_{low}$, $R1_{est}$ and $R1_{high}$, optimizer fixes T_{low}, T_{est} and T_{high} values as 7.000, 15.000 and 33.000 respectively. Single point placement strategy places the operator on site S_2 since estimated |R2| > |R1| +|T| at compile time whereas robust placement strategy places the operator on site S_1 since S_1 is the robust site for the values of T_{low}, T_{est} and T_{high} meaning for any three values of T, in other words the response time does not exceed the minimum response time with threshold. Performance study shows that response time with single point placement is kept at minimum whereas with robust placement the response time is increased by 2,5% to 4,2% which is in accepted limit for the error in the range of [-80, 40]. On the other hand, for the range of [40, 160], the speed up achieved by robust placement is almost two times. Hence maximum loss in response time of 4,2% gives the chance of performance gain by 200%.

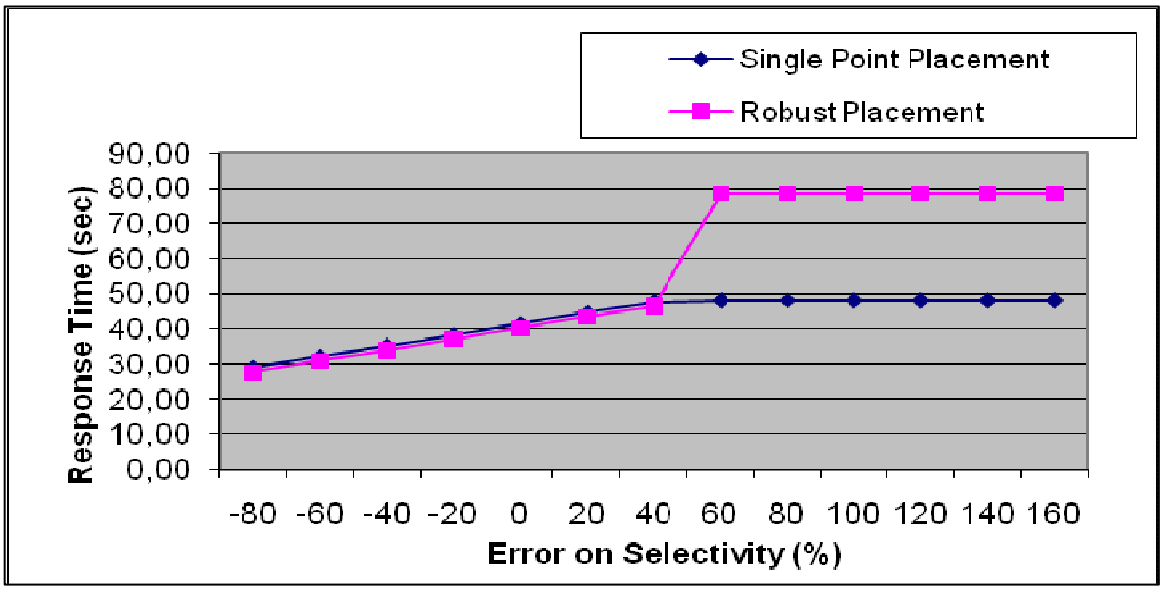

Figure 4.13: Response time with respect to error on selectivity

3.2 Multi-join

Let us consider four relations R1, R2, R3 and R4 residing respectively on sites S_1, S_2, S_3 and S_4 with number of tuples estimated as 10.000, 20.000, 20.000 and 25.000 respectively. We assume a query T = J3(J1(R1,R2), J2(R3,R4)) where the execution plan is a bushy tree. The query result is expected on the site S_5. The migration spaces of joins are MS_{J1}= {S_1, S_2, S_5} and MS_{J2}= {S_3, S_4, S_5} and MS_{J3}= {S_1, S_2, S_3, S_4, S_5}. The selectivity factors of joins J1, J2 and J3 are $1.5/\max(\|R_i\|, \|R_j\|)$.

The plan generated by the single-point optimizer is to place J1 on the site S_2 since $|R_2| > |R_1|$, J2 on the site S_4 since $|R_4| > |R_3|$. On the other hand, robust placement maps the J1 onto the site S_1 regarding the values of $R1_{low}$, $R1_{est}$ and $R1_{high}$ which are estimated by the optimizer as 256 KB, 1 280 KB and 3 328 KB respectively. J2 is placed on S_3 by checking the robustness of the site for the values of $R3_{low}$, $R3_{est}$ and $R3_{high}$ given as 512 KB, 2 560 KB and 7 168 KB respectively. Placement of J3 is done on S_4 by both methods. In order to see the impact of the placement method on the response time of the query we compared both methods in the cases of error on the size of the relations in the query; R1, R2, R3 and R4. We generated error on the size of the relation in a range [-80%, 160%] as in the case of single join

Figure 4.14 demonstrates the impact of the error on |R1| on response time of the query. For the error in the range [-80%, 80%] MJOs placed by single-point placement method remain on their initial sites and the query response time is kept at minimum. On the other hand, if we look at the performance of robust placement method for the same range of error we find that the query response time is higher by [2,9%, 3,43%] since there is mobility overhead caused by J1 and J2. The overhead of each join is less than the threshold. For the range of [80%, 160%], first join placed by robust placement decides to stay on the site S_1 whereas it decides to migrate to S_1 by single-point placement. We remark that robust placement method causes speedup from 16,8% to 36,4% corresponding to the range of [80%, 160%].

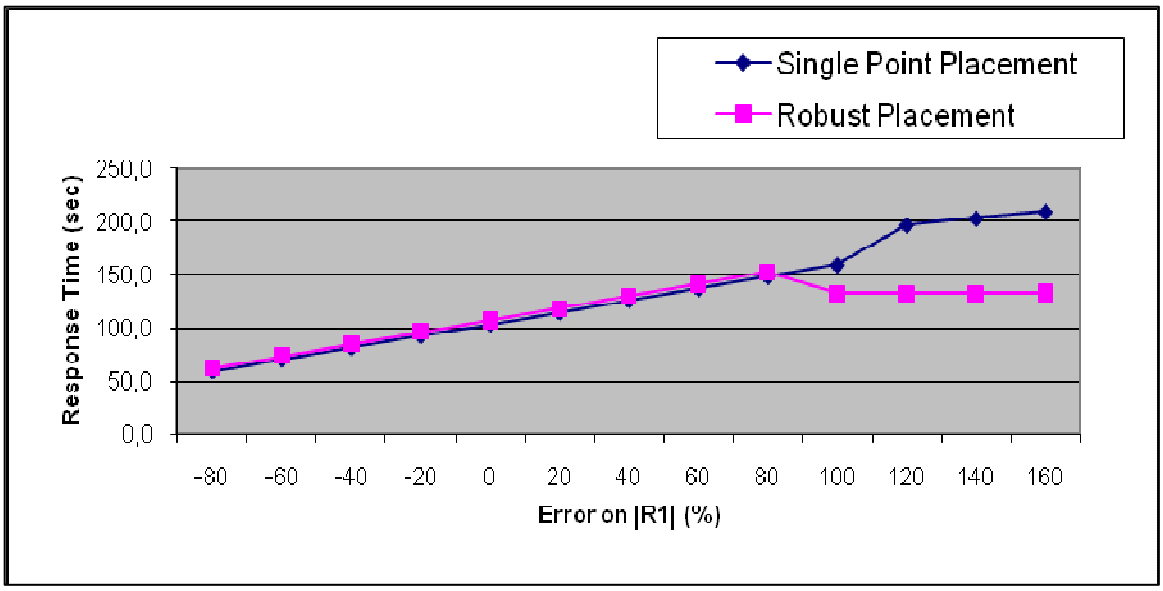

Figure 4.14: Response time with respect to error on |R1|

Figure 4.15 demonstrates the impact of the error on |R2| estimated at compile-time on the query response time with the two placement methods. Response time achieved by robust placement is higher by [3,4%, 4,4%]. Changing size of the second relation does not affect the response time of the query in both methods. Since the MJOs do not take any adaptation decision depending on R2. MJOs placed by single-point placement method remain on their initial sites whereas the first and second MJOs placed by robust placement method move to the sites S_2 and S_4 respectively. Performance of robust placement method is higher than

single-point placement since the response times of the first and second MJOs include mobility overhead as well.

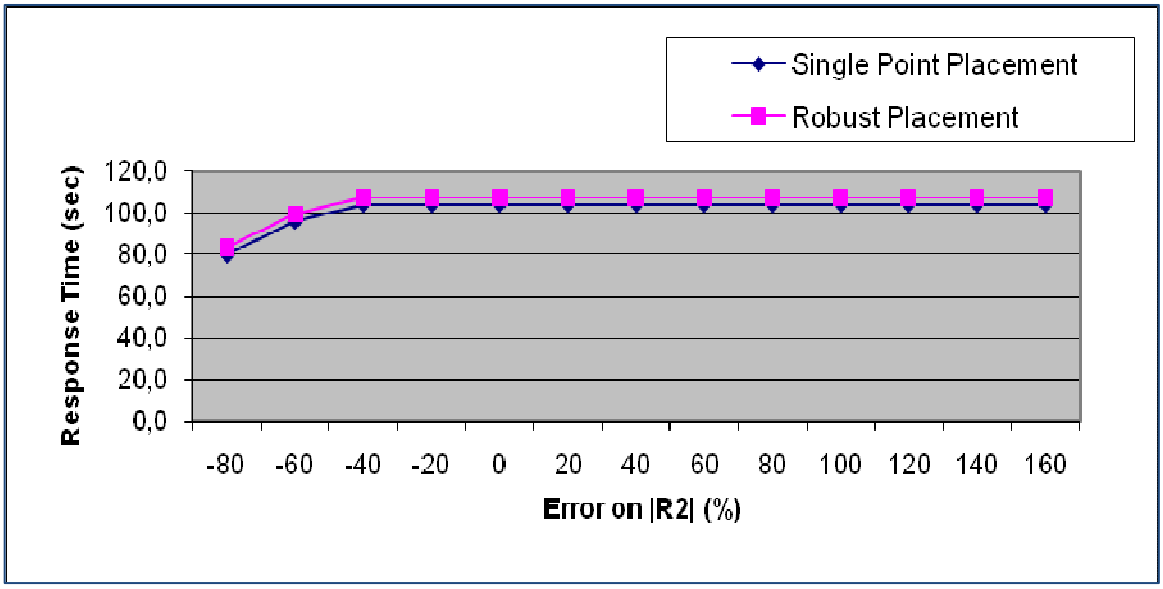

Figure 4.15: Response time with respect to error on |R2|

Figure 4.16 shows the impact of the error on |R3| on the query response time. For the range [-80%, 20%], MJOs placed by single-point placement method remain on their initial sites and the query response time is at minimum. For the same range of error, response time of the single-point placement is higher by [2,9%, 5,6%]. On the other hand, for the range [20%, 160%], single-point placement causes an unexpected increase in response time since J2 decides to migrate to site S_3. If we look at the performance of robust placement method, for the range of [-80%, 20%] we find speedup by [4,6%, 57,97%] since J3 decides to stay on site S_3.

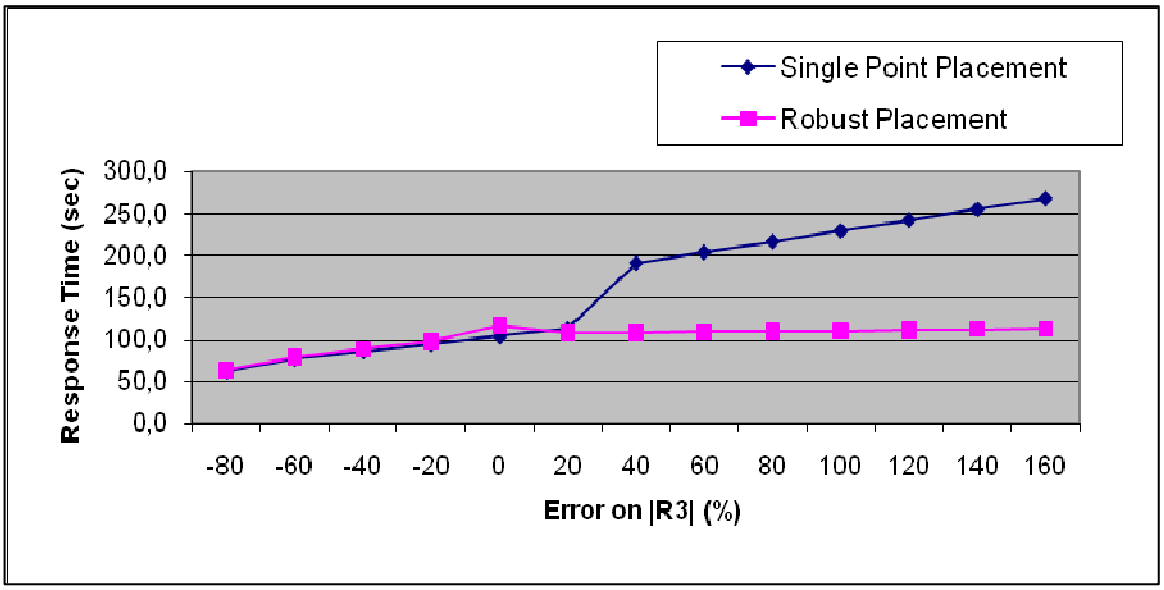

Figure 4.16: Response time with respect to error on |R3|

Figure 4.17 demonstrates the impact of the error on |R4| on the query response time with the two placement methods. Performance of robust placement method is less than the single-point placement by [4,1%, 5,6%]. Again, like the case shown in Figure 4.15, changing |R4| does not make major change in the query response time in both methods since the MJOs do not take any adaptation decision depending on the |R4|.

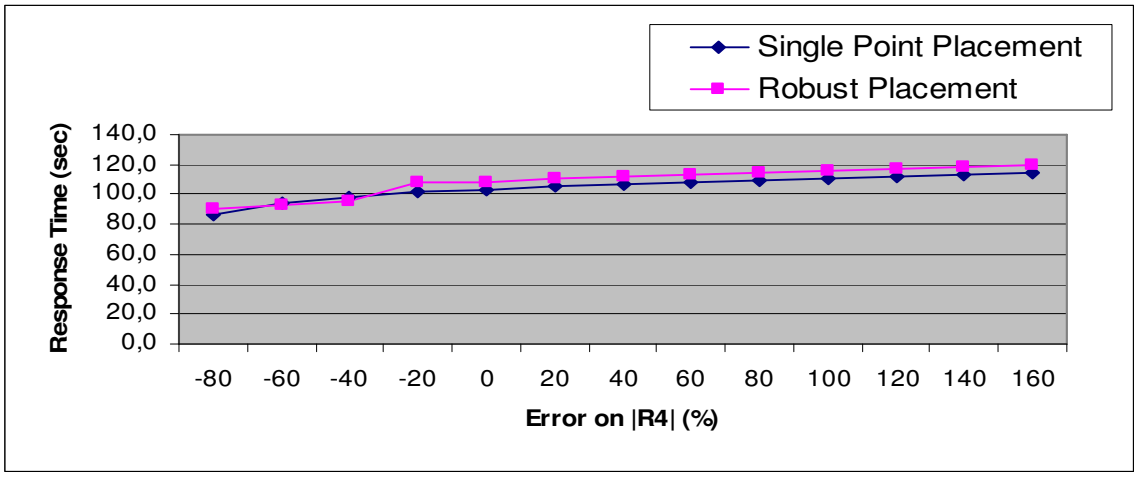

Figure 4.17: Response time with respect to error on |R4|

3.3 Discussion on the Performance Evaluation

In this section, we analyzed the performance of single-point placement and robust placement methods when there is error on the compile-time estimation.

In the first part, we analyzed the response time of a single join placed by single-point and robust placement methods when there is an error on |R1| over a range [-80%, 160%]. The result of the study showed that robust placement method outperformed by almost up to a factor of three when the error on |R1| exceeds 20%. For the error less than 20%, the robust placement performance is higher than single-point placement by [2,9%, 3.7%]. This loss is less then the threshold of mobility overhead which is 6%.

In the second part, we analyzed the response time of an execution plan composed of three joins in the form of a bushy tree with the MJOs placed by the single-point and robust placement methods. Again, the error generated on |R1|, |R2|, |R3| and |R4| is between [-80%, 160%]. Robust placement has better performance in case of the error on |R1| when the error exceeds 80% and in case of the error on |R3| when the error exceeds 20%. The performance gain reaches up to 60% in response time. Performance loss with the robust placement in cases of error on |R2| and |R4| estimated at compile-time does not exceed threshold (e.g. 5,6% maximum). We noticed that the error on the size estimated at compile-time of the first relation has greater impact on the query response time. Robust placement method outperforms when there is error on the size of the first relation since it expects adaptation and chooses the initial site according to this possibility.

4. Conclusion

In this chapter first, we have demonstrated the performance evaluation of mobile query operators specifically developed for restricted sources. The performance study has shown that the mobility was effective (i) when the variation between the compile-time estimated and run-time computed sizes of the data R1and R2', is more than 30%, (ii) the network bandwidth is less than 640 Kb/s, (iii) the processor frequency is less than 550 MHz, and (iv) SMDJoin has evidently better performance than MDJoin for the cases where there is a variation between the compile-time estimated and run-time computed size of R2'. In large-scale data integration

where data sources are expected to be heterogeneous and autonomous; the various data sources do not export needed statistical information or the statistical information is subject to be obsolete. It is difficult for an optimizer to estimate the cost, thus 30 % of error in the estimation is not high. In addition, a network bandwidth of 640 Kb/s can often be experienced. So in this context, the contribution of mobility can be taken into consideration. MDJoin can be efficient for the cases where the probability of estimation error on data coming from the second source is low whereas the second operator, SMDJoin is efficient for the cases where the probability of estimation error on data coming from the second source is high.

Second work presented the performance evaluation study on robust placement method to map, at compile-time, the mobile relational operators (MROs) of a query on execution sites over a large scale environment. The study focused on comparing the performance of mobile join operators (MJOs) of a query execution plan by single-point and robust placement methods in cases of estimation errors. For the robust placement method, we assumed to have an interval defined by the low, average (est) and high estimation points computed at compile-time related with the size of relations. Single-point placement method mapped the MJOs onto execution sites by finding the optimal performance based on average estimation, whereas the robust placement tried to place them by leaning on low, average and high estimations. We discovered that with a risk of loosing around 6% in response time, it is possible to gain up to 300% in some cases with the robust placement method.

Chapter V Conclusion and Future Work

The main objective of data integration systems based on mediator-wrapper architecture is to provide uniform and transparent access to diverse data sources which are scattered over large scale network. The characteristics of the data sources (distribution, heterogeneity and autonomy), the environment (high transfer costs and presence of unpredictable events) and the changing user requirements (like accuracy, completeness, consistency besides minimum response time) in data integration systems pose various problems in terms query processing. Traditional query processing methods have to face challenges in terms of expressive power, easy addition of a new source, effective and efficient reformulation of the query for local data sources, query optimization in the presence of autonomous data sources, changing run-time conditions and access restrictions of the sources.

Although the spectrum of challenges is large, in this thesis our challenge was to focus on query optimization in the presence of restricted sources over large scale network. Regarding our objective there are three main axis of research: (1) works related with **limited query capabilities** [DUS 98, LI 03, MAN 01, VAS 97 and YER 99] of the sources due to security and performance reasons, (2) works concentrating on **cost models** [ADA 96, AMB 98, GAR 96, NAA 98 and ZHU 03] aiming to make good estimation at compile time overcoming the difficulty of undependable statistics of the data sources, sources not exporting their cost models and changing run-time conditions and (3) works on **adaptive query optimization** methods where the challenge is to make re-optimization at run-time in order to handle sub-optimal execution plans [AMS 97, AVN 00, KAB 98 and ZHO 05].

Our comments on first group of work regarding the query optimization in the presence of **restricted sources** can be summarized as (1) the capabilities of restricted sources are

modeled and by binding patterns, free grammars, operator model and common ontologies, (2) traditional set of relational operators is not enough for query execution in the presence of restricted sources: new operators capable to handle access restrictions are required and (3) optimization algorithms are necessary for deciding how a query can be answered by given capabilities of the sources at a minimum cost.

Our findings related with second group of work focusing on the development of **cost models** can be summarized as (1) although different cost models like historical, calibrating and generic are proposed in data integration systems, it is difficult is to make good cost estimation at compile time capturing necessary information from different autonomous data sources and (2) occurrence of the unpredictable events of execution time can make execution plans sub-optimal at run-time. With the objective of overcoming these drawbacks, large amount of research were carried out in the area of adaptive query optimization where the idea is to change execution plan at run time with better knowledge of data and run-time conditions.

Our remarks on **adaptive query optimization** methods can be stated as (1) large number of adaptive methods with different type of modification (e.g. re-optimization, re-scheduling, replacement and dynamic operators), place of modification (e.g. inter-operator or intra-operator), type of control (e.g., centralized or decentralized), type of triggering event (e.g. estimation errors on statistics, memory unavailability, delays in data arrival, user preferences) and type of re-optimization (e.g. reactive or proactive) exist, (2) centralization of control on monitoring the run time conditions and making the decision about modification makes adaptive query optimization methods inapplicable over large scale network due to the risk of creating bottleneck in the presence of large amount of data and control message passing towards the central authority and (3) modification of the plan should be done with the concern of reducing transfer costs since in large scale network there is high network latency and low bandwidth. Adaptive query optimization methods with decentralized control (monitoring run-time parameters and making the decision of adaptation at operator level or sub-query level) is the solution for the problem indicated at (2) [COL 04, JON 03, IVE 99 and URH 00].

Decentralized adaptive query optimization methods are dynamic operators at operator level, communicating agents or brokers at sub-query level which try to improve the cost of local processing in case of inaccurate estimates. Mobile query execution model is another example of decentralized adaptive query optimization methods where the aim of adaptation is to reduce cost of data transfer [ARC 04, HUS 05, MOR 03 and OZA 05a]. In this model relational operators are based on mobile agents: they can migrate from one site to another among the sites determined by the optimizer according to the system state, data availability and run-time calculation of statistics on data. Mobile query execution model is a promising method for facing the challenges mentioned in (2) and (3) of the previous paragraph. But we have discovered out that mobile query execution model is inapplicable with the sources posing access restrictions and have no link with the optimizer in mapping the mobile relational operators to the sites regarding their sensitivity to initial placement. These two open aspects related with mobile query execution model were the motivation of works presented in this thesis.

In our first work, we proposed **mobile join operators designed for restricted sources** in order to extend the mobile query execution model in a way to handle restricted sources in query execution. In this perspective, we designed two mobile operators, Mobile Dependent Join (MDJoin) and Sampling Dependent Join (SMDJoin). They are able to adapt to run time parameters by changing their place of execution and able to work with access restrictions imposed by the sources. The difference between the two new query operators lies in their level of adaptation ability to the execution environment: SMDJoin has an additional sampling step in order to make better estimation about the second source as well. In our second work, we addressed the problem of **initial placement** taking into account the adaptation capability of mobile relational operators at run-time (mobile join, MDJoin and SMDJoin). Our challenge was not finding good placement based on single-point estimates but defining acceptable placement for an estimation interval in order to avoid dramatic performance at run-time.

The **performance evaluation study** given in this thesis is on our proposed methods: mobile relational operators for restricted sources and initial placement of mobile relational operators. We had done our performance evaluation on both of the methods in a simulation environment. First study evaluated the performances MDJoin and SMDJoin in comparison to

the dependent join operator of the standard strategy. The concern of the evaluation was on their performance with respect to the following parameters: response time in the presence of the variations between the compile-time estimations and the run-time computations of the sizes of the relations, speedup[4] realized in comparison with the network bandwidth and CPU frequency. The second study compared single-point and robust placement methods mapping single join and multi-join. Both methods were tested when there is error on the estimations related to the sizes of relations.

In order to wrap-up; the **contribution** of the thesis is as follows; (1) two mobile join operators (MDJoin and SMDJoin) which are capable to work with the sources having limited query capabilities and yet capable to adapt to run time conditions when there is estimation error and (2) initial placement method which finds robust site to place mobile relational operators taking into account their migration (adaptation) possibility. Performance study on mobile relational operators for restricted sources showed that if the size of the data from the first source is more than 30% different than the estimation made by the optimizer at compile time, our mobile dependent operators have better performance compared to the operator of the standard strategy. Against the variations of the size of the data coming from the second source SMDJoin is superior to MDJoin, since due to its sampling step it sees the variation of the size of the data of the second source. The overhead of this sampling step for SMDJoin is less than 10 % of increase in response time compared to MDJoin. The performance study on initial placement of mobile relational operators showed that robust placement method has better performance in case of the error on the size of the first relation exceeds 80% (under-estimation of the size at compile time). We noticed that the error on the size estimated at compile-time of the first relation has greater impact on the query response time. Robust placement method outperforms when there is error on the size of the first relation since it expects adaptation and chooses the initial site according to this possibility.

As a future work we want to extend our simulation model as it would allow us to measure the impact of parameters like data availability and system state on the decision

[4] The speedup: response time of a DJoin / response time of a mobile join operator

function of the mobile operators in the presence of our proposed methods. Another interesting perspective can be to extent our initial placement method for mobile relational operators in two directions: introduction of a method to minimize the **effects of the estimation errors** in the query graph on response time and proposition of a method for **multi-query placement**.

In the context of the research on the **effect of estimation errors** on the response time of the query we want to propose a method which (1) measures uncertainty in estimation and defines the propagation rule in the query graph (2) captures multiple estimation errors in defining interval for each mobile relational operator.

For the perspective of definition of an initial placement method for mobile relational operators in **multi-query optimization** we want to focus on query distribution in case of data replication and inter site joins trying to maximize the parallelism and increase the efficiency.

References

[ADA 96] S. ADALI, K.S. CANDAN, Y. PAPAKONSTANTINOU and V.S. SUBRAHMANIAN; *Query Caching and Optimization in Distributed Mediator Systems,* Proc. of the SIGMOD Conf., Montreal, 4-6 June 1996, pp. 137-148.

[AMB 98] J.L. AMBITE and C.A. KNOBLOCK; *Flexible and Scalable Query Planning in Distributed and Heterogeneous Environments,* Proc. of the Fourth International Conference on Artificial Intelligence Planning Systems, Pitsburg, USA, June 1998.

[AMS 97] L. AMSALEG, M. FRANKLIN and A. TOMASIC; *Dynamic Query Operator Scheduling for Wide-Area Remote Access*, INRIA, France, Technical Report No: 3283, October 1997.

[ARC 04] J.P. ARCANGELI, A. HAMEURLAIN, F. MIGEON and F. MORVAN; *Mobile Agent Based Self-Adaptive Join for Wide-Area Distributed Query Processing,* Journal of Database Management, 15(4), Oct-Dec 2004, pp. 25-44.

[AVN 00] R.AVNUR and J.M. HELLERSTEIN; *Eddies: Continuously Adaptive Query Processing*, SIGMOD, 2000.

[BAB 05a] S. BABU and P. BIZARRO; *Adaptive Query Processing in the Looking Glass*, Proc. of Second Biennal Conference on Innovative Data Systems Research (CIDR), January 2005.

[BAB 05b] S. BABU, P. BIZARRO and D. DeWITT; *Proactive Re-Optimization,* ACM International Conference on Management of Data (SIGMOD), 2005.

[BAR 01] BarnesAndNoble.com web site, hhtp://www.barnesandnoble.com, March 2001.

[BAY 97] R.J. BAYARDO et al.; *Infosleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments*, In ACM SIGMOD Record, vol. 26, No. 2, 1997.

[BIZ 05] P. BIZARRO, S. BABU and D. DeWITT, J. WIDOM; *Content-Based Routing: Different Plans for Different Data*, Proc. of the 31st VLDB Conference, Trondheim, Norway, 2005.

[BOU 01] L. BOUGANIM, F. FABRET, F. PORTO and P. VALDURIEZ; *Processing queries with expensive functions and large objects in distributed mediator systems,* ICDE 2001.

[CHA 93] S. CHAUDHURI and K. SHIM; *Query optimization in the presence of foreign functions*, Proc. of the VLDB, 1993.

[CHE 92] M-S. CHEN, P.S. YU and K-L. WU; *Scheduling and Processor Allocation for Parallel Execution of Multi-Join Queries,* 8th International Conference on Data Engineering, Feb.2-3, Tempe Arizona, 1992.

[CHE 95] C. CHEKURI, W. HASAN and R. MOTWANI; *Scheduling Problems in Parallel Query Optimization,* Symposium on Principles of Database Systems PODS, ACM Press, 1995, pp. 255-265.

[COL 04] C.COLET and T.T. VU; QBF: *A Query Broker Framework for Adaptable Query Evaluation*, Proc. of 6th International Conference on Flexible Query Answering Systems, Verlag Publishers, Lyon, France, June 2004.

[DU 92] W. DU, R. KRISHNAMURTHY and M.-C. SHAN ; *Query Optimization in a Heterogeneous DBMS,* Proc. of the 18th International Conference on VLDB, Vancouver, 23-27 Aug. 1992, pp.277-291.

[DUS 98] O. M. DUSCHKA; *Query Planning and Optimization in Information Integration*. Ph.D thesis, Stanford University, Stanford, Calfornia, 1998.

[ERG 07] B. ERGENÇ (ÖZAKAR), F. MORVAN and A. HAMEURLAIN; *Robust Placement of Mobile Relational Operators for Large Scale Distributed Query Optimization,* Proc. of the PDCAT07, Adelaide, Australia, 2007.

[FRA 98] M.J. FRANKLIN, B.T. JONSSON and D. KOSSMANN; *Performance Tradeoffs for* Client-Server Query Processing, SIGMOD Conference, 1998, pp. 9-18.

[GAR 95] G. GARDARIN, J.R. GRUSER and Z.H. TANG; *A Cost Model for Clustered Object Oriented Databases*, Modern Database Systems: The Object Model, Interoperability, and Beyond. ACM Press, 1995.

[GAR 96] G. GARDARIN, F. SHA and Z.-H. TANG; *Calibrating the Query Optimizer Cost Model of IRO-DB, An Object-Oriented Federated Database System*, Proc. of the 22nd Intl. Conf. on VLDB, Bombay, 3-6 Sept.1996, pp. 378-389.

[GAR 96b] M.N. GAROFALAKIS and Y.E. IOANNIDIS; *Multi-Dimensional Resource Scheduling for Parallel Queries*, SIGMOD, Montreal, Canada, 1996.

[GEN 97] M.R. GENESERETH, A.M. KELLER and O. DUSCKA; *Infomaster: An Information Integration System,* Proc. of ACM SIGMOD Conference*,* May 1997.

[GOL 89] D.E. GOLDBERG; *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[GOL 00] R. GOLDMAN and J. WIDOM; *WSQ/DSQ: A Practical Approach for Combined Querying of Databases and the Web.* Proc. of ACM SIGMOD Conf., 2000.

[GRA 97] P.D.M. GRAY et al.; *KRAFT: Knowledge Fusion from Distributed Databases and Knowledge Bases*, Proc. of the DEXA 1997, Toulouse, France, 1997, pp. 682-691.

[GUP 97] H. GUPTA, V. HARINARAYAN, A. RAJARAMAN and J. ULLMAN; *Index Selection for OLAP*, P*roc. of the International Conference on Data Engineering*, Binghamton, UK, April, 1997.

[HAA 97] L.M. HAAS, D.KOSSMANN, E.L. WIMMERS and J.Y. YANG; *Optimizing Queries Across Diverse Data Sources*, Proc. of the VLDB Conference, 1997, pp.276-285.

[HAG 00] D. HAGIMONT and L. ISMAIL; *Agents Mobiles et Client/Serveur: Evaluation de Performance et Comparison,* Technique et Science Informatiques, 19(9), Hermes, 2000, pp.1223-1244.

[HAL 01] A.Y. HALEVY; *Answering Queries Using Views: A Survey*, VLDB Journal, 2001, pp. 270-294.

[HAN 07] W-S. HAN, J. NG, V. MARKL, H. KACHE and M. KANDIL; Progressive Optimization in a Shared-Nothing Parallel Database, SIGMOD'07, Beijing, China, 2007.

[HAS 94] W. HASAN and R. MOTWANI; *Optimization Algorithms for Exploiting the Parallelism-Communication Tradeoff in Pipelined Parallelism*, Proceedings of the 20th VLDB Conference Santiago, Chile, 1994.

[HAS 00] W. HASSELBRING; *Information System Integration*, Communications of the ACM, vol. 43, June 2000, pp. 32-38.

[HEL 00] J.M. HELLERSTEIN, M.J. FRANKLIN, S. CHNADRASEKARAN, A. DESHPANDE, K. HILDRUM, S. MADDEN, V. RAMANA and M.A. SHAH; *Adaptive Query Processing: Technology in Evolution*, IEEE Data Engineering Bulletin, vol. 23, no. 2, 2000.

[HON 91] W. HONG and M. STONEBRAKER; *Optimization of Parallel Query Execution Plans in XPRS,* Proc. of the First International Conference on Parallel and Distributed Information Systems, December, 1991.

[HUS 05] M. HUSSEIN, F. MORVAN and A. HAMEURLAIN; *Embedded Cost Model in Mobile Agents for Large Scale Query Optimization*, Intl. Symposium on Parallel and Distributed Computing, 2005, pp. 199-206.

[HUS 06] M. HUSSEIN, F. MORVAN and A. HAMEURLAIN; *Dynamic Query Optimization: From Centralized to Decentralized*, 19th International Conference on Parallel and Distributed Computing Systems, San Francisco, California, International Society for Computers and their Applications (ISCA), September 2006, pp. 273-279.

[IOA 87] Y.E. IOANNIDIS and E. WONG; *Query Optimization by Simulated Annealing*, Proc. of ACM STGMOD Conf., 1987.

[IOA 91] Y. E. IOANNIDIS and S. CHRISTODOULAKIS; *On the Propagation of Errors in the Size of Join Results,* ACM SIGMOD, 1991, pp. 268-277.

[IOA 92] Y.E. IOANNIDIS et al ; *Parametric Query Optimization,* Proc. of the International Conference on Very Large Databases, August, 1992.

[IVE 99] Z. IVES, D. FLORESCU, M. FRIEDMAN, A. LEVY and D. WELD; *An Adaptive Data Integration System for Data Integration,* Proceedings of the ACM SIGMOD International Conference on Management of Data, June 1999, pp. 299-310.

[IVE 04] Z. IVES, A. HALEVY and D. WELD; *Adapting to Source Properties in Processing Data Integration Queries,* Proc. of ACM SIGMOD International Conference on Management of Data, June, 2004.

[JON 03] R. JONES and J. BROWN; *Distributed Query Processing via Mobile Agents*. University of Maryland, from http://www.cs.umd.edu/~keleher/818s97/ryan/paper.html, Retrieved October 19, 2003.

[KAB 98] N. KABRA and D. DeWITT; *Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans*, Proc. of ACM SIGMOD, 1998, pp. 106-117.

[KAC 06] H.KACHE, W_S. HAN, V. MARKL, S. EWEN and V. RAMAN; POP/FED: Progressive Query Optimization for Federated Queries in DB2, VLDB, Korea, 2006.

[KHA 00] L. KHAN, D. MC LEOD and C. SHAHABI; *An Adaptive Probe-Based Technique to Optimize Join Queries in Distributed Internet Databases*. Knowledge and Information Systems, vol. 2, 2000, pp. 373-385.

[KIR 95] T. KIRK, A.Y. LEVY, Y. SAGIV and D. SRIVASTAVA; *The Information Manifold*, In AAAI Spring Symposium on Information Gathering, 1995.

[KOW 88] R.A. KOWALSKI; *The Early Years of Logic Programming*, Communications of ACM, vol.31, 1988. pp.38-43.

[LAM 04] E. LAMBRECHT, S. KAMBHAMPATI and S. GNANAPRAKASAM ; *Optimizing Recursive Information Gathering Plans*. Journal of Intelligent Information Systems, vol. 22(2), March 2004, pp.119-153.

[LAN 91] R.-S.-G. LANZELOTTE AND P. VALDURIEZ ; *Extending the Search Strategy in a Query Optimizer*, Proc. of the 17th International Conference on Very Large Data Bases, Morgan Kaufmann, Barcelona, Spain, September 1991, pp. 363-373.

[LEN 02] M. LENZERINI; *Data Integration: A Theoretical Perspective*, PODS, 2002, pp. 243-246.

[LEV 96] A.Y. LEVY, A. RAJARAMAN and J.J. ORDILLE; *Querying Heterogeneous Information Sources Using Source Descriptions,* VLDB, 1996, pp. 251-262.

[LI 03] C. LI; *Computing Complete Answers to Queries in the Presence of Limited Access Patterns*, The VLDB Journal, Vol. 12, 2003, pp. 211-227.

[LIU 98] L. LIU, C. PU and K. RICHINE ; *Distributed Query Scheduling Service: An Architecture and Its Implementation*, Special issue on Compound Information Services, International Journal of Cooperative Information Systems (IJCIS), vol.7, No.2&3, 1998, pp. 123-166.

[LIU 05] B. LIU and E.A. RUNDENSTEINER; *Revisiting Pipelined Parallelism in Multi-Join Query Processing,* Proc.of 31st VLDB Conference, Norway, 2005, pp. 829-843.

[LO 93] M-L. LO, M-S. CHEN, C.V. RAVISHANKAR and P.S. YU; *On Optimal Processor Allocation to Support Pipelined Hash Joins,* Proc. of the ACM SIGMOD International Conference on Management of Data, Washington, D.C., United States, 1993, pp. 69-78.

[MAC 86] L.F. MACKERT and G.M. LOHMAN; *R* Optimizer Validation and Performance Evaluation for Distributed Queries,* Proc. of the 12th Int. Conf. On VLDB, 1986 pp. 149-159.

[MAN 01] I. MANOLESCU; *Optimization Techniques for Integration of Heterogeneous and Distributed Data Sources*. Ph.D thesis, Universty of Versailles, 2001.

[MAN 02] I. MANOLESCU, L. BOUGANIM, F. FABRET and E. SIMON; *Efficient Querying of Distributed Resources in Mediator Systems.* Proc. of the Confederated International Conferences DOA, CoopIS and ODBASE, LNCS 2519, Springer-Verlag, 2002, pp. 468 – 485.

[MAR 04] V. MARKL, V.RAMAN, D. SIMMEN, G. LOHMAN, H.PİRAHESH and M. CILIMDZIC; *Robust Query Processing through Progressive Optimization*, SIGMOD, Paris, Frabce, 2004.

[MEN 98] E. MENA et al.; *Domain Specific Ontologies for Semantic Information Brokering on The Global Information Infrastructure*, Proc. of the International Conference on Formal Information Sytems, Trento, Italy, 1998.

[MOR 88] K.A. MORRIS; *An Algorithm for Ordering Subgoals in NAIL!,* Proc. of the Symposium on Principles of Database Systems (PODS), Chicago, Illinois, 1988, pp. 82-88.

[MOR 02] F. MORVAN and A. HAMEURLAIN; *Dynamic Memory Allocation Strategies for Parallel Query Execution,* Proc. of the 17th ACM Symposium on Applied, 2002, pp. 897-901.

[MOR 03] F. MORVAN, M. HUSSEIN and A. HAMEURLAIN; *Mobile Agent Cooperation Methods for Large Scale Distributed Dynamic Query Optimization,* Fifth International Workshop on Parallel and Distributed Databases: Innovative Applications and New Architectures (PaDD03), Prague, 1-5 September, 2003.

[NAA 98] H. NAACKE, G. GARDARIN and A. TOMASIC*; Leveraging Mediator Cost Models with Heterogeneous Data Sources*, ICDE, Orlando, 23-27 Feb. 1998, pp. 351-360.

[NAA 99] H. NAACKE ; *Modèles de coût pour médiateurs de bases de données hétérogènes*, Thèse de Doctorat, Spécialité Informatique, Université de Versailles Saint-Quentin-en-Yvlenies, France, 1999.

[NIE 01] Z. NIE and S. KAMBHAMPATI; *Joint Optimization Of Cost and Coverage of Query Plans in Data Integration, Proc. of CIKM, 2001, pp. 223-230.*

[OZA 05a] B. ÖZAKAR, F. MORVAN and A. HAMEURLAIN; *Query Optimization: Mobile Agents versus Accuracy of The Cost Estimation*, International Journal of Computer Systems Science and Engineering, vol. 20, number 3, May 2005, pp: 161-168.

[OZA 05b] B. ÖZAKAR, F. MORVAN and A. HAMEURLAIN; *Mobile Join Operators for Restricted Sources,* International Journal on Mobile Information Systems 1, IOS Press, 2005.

[OZS 99] T. OZSU and P. VALDURIEZ; *Principles Of Distributed Database Systems*, Prentice Hall, 1999.

[PAP 95] Y. PAPAKONSTANTINO, H. GARCIA-MOLINA and J. WIDOM: *Object Exchange across Heterogeneous Information Sources*, Proc. of Data Engineering, Tapei, Taiwan, 1995, pp. 251-260.

[POT 01] R. POTTINGER and A. HALEVY; *Minicon: A Scalable Algorithm for Answering Queries Using Views*, VLDB 2001, pp. 182-198.

[RAJ 95] A. RAJARAMAN and Y. SAGIV, J. ULLMAN; *Answering Queries Using Templates with Binding Patterns*. Proc. of ACM PODS, San Jose, CA, 1995.

[ROD 00] M. RODRIQUEZ-MARTINEZ and N. ROUSSOPOULOS; *MOCHA; A Self-Extensible Middleware System for Distributed Data Sources,* Proc. of ACM SIGMOD Conference, Dallas, Texas, USA, May 2000.

[ROT 96] M.T. ROTH, M. ARYA, L. HAAS, M. CAREY, W. CODY, R. FAGIN, P. SCHWARZ, J. THOMAS and E. WIMMERS; *The Garlic Project*, Proc. of ACM SIGMOD Int. Conference on Management of Data, 1996.

[ROT 99] M.T. ROTH, F. OZCAN and LM. HAAS; *Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System*, Proc. of the 25th Very Large Data Bases (VLDB), 1999, pp. 590-610.

[SCH 90] D.A. SCHNEIDER and D.J. DEWITT; *Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines*. Proc. of the 16th VLDB Conference Brisbane, Australia, 1990.

[SEL 79] P. SELINGER, M. ASTRAHAN, D. CHAMBERLIN, R. LORIE and T. PRICE; *Access Path Selection in a Relational Database System*, Proc. of ACM SIGMOD Conference on Management of Data, Boston, Massachusetts, 1979, pp. 23-34.

[SHA 93] M.-C. SHAN; *Pegasus Architecture and Design Principles*, Proc. of the ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, June 1993.

[SHE 93] E.J. SHEKITA, H.C. YOUNG and K.L. TAN; *Multi-Join Optimization for Symmetric Multiprocessors,* Proc. of the 19th International Conference on VLDB, 1993, pp. 479-492.

[SRI 06] U. SRIVASTAVA, K. MUNGALA, J. WIDOM and R. MOTWANI; *Query Optimization over Web Services*, VLDB, Seoul, Korea, September 2006.

[SWA 88] A.-N. SWAMI and A. GUPTA ; *Optimization of Large Join Queries*, Proc. of the ACM SIGMOD International Conference on Management of Data, ACM Press, Chicago, Illinois, USA, Juin 1988, pp. 8-17.

[TOM 98] A. TOMASIC, L. RASCHID and P. VALDURIEZ; *Scaling Access to Distributed Heterogeneous Data Sources with DISCO*, IEEE Transactions on Knowledge and Data Engineering, 1998.

[ULL 97] J. D. ULLMAN; *Information Integration Using Logical Views*, ICDT, 1997, pp. 19-40.

[URH 00] T. URHAN and M. FRANKLIN; *Xjoin: A Reactively Scheduled Pipelined Join Operator*, IEEE Data Engineering Bulletin, IEEE CS, vol. 23, N. 2, 2000.

[VAS 97] V. VASSALOS and Y. PAPAKONSTANTINOU; *Describing and using query capabilities of heterogeneous sources,* Proc. of VLDB, pp. 256-265, 1997.

[WID 95] J. WIDOM; *Research Problems in Data Warehousing*, Proc. of the 4th Int'l Conference on Information and Knowledge Management (CIKM*)*, November 1995.

[WIE 92] G. WIEDERHOLD; *Mediators In the Architecture of Future Information Systems*, IEEE Computer 25(3), 1992, pp. 38-49.

[XML 01] XML parser for Java; Available at http://www.alphaworks.ibm.com/tech/xml4j, 2001.

[YER 99] R. YERNEI, C. LI, H. GARCIA-MOLINA and J. ULLMAN; *Computing Capabilities of Mediators*, Proc. of ACM SIGMOD Conference on Management of Data, 1999.

[ZHO 05] Y. ZHOU, B.C. OOI, K-L. TAN and W.H. TOK; *An Adaptable Distributed Query Processing Architecture.* Data and Knowledge Engineering, 53, 2005, pp. 283-309.

[ZHU 98] Q. ZHU and P.-Å. LARSON ; *Solving Local Cost Estimation Problem for Global Query Optimization in Multidatabase Systems*, Journal of Distributed and

Parallel Databases, Kluwer Academic Publishers, vol. 6, n°4, October 1998, pp. 373-421.

[ZHU 00] Q. ZHU, Y. SUN and S. MOTHERAMGARI ; *Developing Cost Models with Qualitative Variables for Dynamic Multidatabase Environments*, Proc. of the 16th International Conference on Data Engineering, IEEE Computer Society, San Diego, California, USA, Mars 2000, pp. 413-424.

[ZHU 03] Q. ZHU, S. MONTHERAMGARI and Y. SUN; *Cost Estimation for Queries Experiencing Multiple Contention States in Dynamic Multidatabase Environments*, Knowledge and Information Systems, vol. 5, No. 1, 2003, pp. 26-49.

[ZHU 04] Y. ZHU, E. RUNDENSTEINER and G. HEINEMAN; *Dynamic Plan Migration for Continuous Queries Over Data Streams.* Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, 2004, pp. 431–442.

Un modèle d'exécution de requêtes mobiles pour des sources à accès restreints en environnement d'intégration de données

Abstract

Query optimization in data integration systems over large scale network, faces the challenges of dealing with autonomous, heterogeneous and distributed data sources, dynamic execution environment and changing user requirements. These issues initiate the need for crafting the traditional optimization methods in a way to produce stable query execution plans, use execution models which are able to adapt to run-time conditions and handle source restrictions. Centralization of the control in adaptive optimization methods result in bottleneck due to large amounts of message passing towards to the site of the central authority over large scale network where network bandwidth is low and network latency is high. In order to overcome this obstacle adaptive query optimization requires decentralized methods. A mobile execution model with mobile relational operators that are able to adapt in an autonomous way and focus on reducing of transfer cost is worth considering in large scale distributed data integration environment. However, this mobile query execution model needs to be extended with new operators to handle source restrictions. In this perspective we propose mobile relational operators developed for restricted sources. Another proposition is related with initial placement of mobile relational operators. The challenge is on defining a placement which would allow acceptable performance instead of a good placement which would cause dramatic performance sometimes at run-time. Finally we present and analyze a performance evaluation on the methods proposed.

Keywords: Data integration, Distribution in large scale, Adaptive optimization, Mobile execution model, Restricted sources, Initial placement, Performance evaluation.

Résumé

L'optimisation de requêtes dans les systèmes d'intégration de données réparties sur un réseau à grande échelle pose des problèmes liés à l'autonomie, l'hétérogénéité et la distribution des sources de données, l'environnement d'exécution dynamique et aux besoins changeant des utilisateurs. Résoudre ces problèmes nécessite de revisiter les méthodes d'optimisation traditionnelles afin de permettre la génération de plans d'exécution stables et d'utiliser des modèles d'exécution capables de s'adapter aux conditions d'exécution et de tenir compte des sources à accès restreints. La centralisation du contrôle dans les méthodes d'optimisation dynamique engendre un goulot d'étranglement dû au nombre important de messages transmis au site contrôlant l'optimisation à travers un réseau à grande échelle (à faible bande passante et à forte latence). Afin de résoudre ce problème, l'optimisation dynamique des requêtes nécessite de décentraliser ces méthodes. Dans un environnement d'intégration de données distribuées à grande échelle, un modèle d'exécution mobile, avec des opérateurs relationnels mobiles capables de s'adapter de façon autonomes et réduisant les coûts de communication a été proposé dans l'équipe. Cependant, ce modèle d'exécution de requêtes mobiles nécessite d'être étendu avec des nouveaux opérateurs afin de tenir compte des sources à accès restreints. Dans cette perspective, nous proposons des opérateurs relationnels mobiles développés pour des sources à accès restreints. Une seconde proposition est liée au placement initial des opérateurs mobiles. Le but est de déterminer un placement permettant d'obtenir des performances acceptables plutôt qu'un bon placement engendrant parfois de très mauvaises performances lors de l'exécution. Enfin, nous évaluons les performances des méthodes proposées.

Mot-Clés : Intégration de données, Répartition à grande échelle, Optimisation dynamique de requêtes, Modèle d'exécution de requêtes mobiles, Sources à accès restreints, Placement initial, Evaluation des performances.