# Evolution of Query Optimization Methods

Abdelkader Hameurlain and Franck Morvan

Institut de Recherche en Informatique de Toulouse IRIT, Paul Sabatier University,
118, Route de Narbonne, 31062 Toulouse Cedex, France
Ph.: 33 (0) 5 61 55 82 48/74 43, Fax: 33 (0) 5 61 55 62 58
hameur@irit.fr, morvan@irit.fr

**Abstract.** Query optimization is the most critical phase in query processing. In this paper, we try to describe synthetically the evolution of query optimization methods from uniprocessor relational database systems to data Grid systems through parallel, distributed and data integration systems. We point out a set of parameters to characterize and compare query optimization methods, mainly: (i) size of the search space, (ii) type of method (static or dynamic), (iii) modification types of execution plans (re-optimization or re-scheduling), (iv) level of modification (intra-operator and/or inter-operator), (v) type of event (estimation errors, delay, user preferences), and (vi) nature of decision-making (centralized or decentralized control).

The major contributions of this paper are: (i) understanding the mechanisms of query optimization methods with respect to the considered environments and their constraints (e.g. parallelism, distribution, heterogeneity, large scale, dynamicity of nodes) (ii) pointing out their main characteristics which allow comparing them, and (iii) the reasons for which proposed methods become very sophisticated.

**Keywords:** Relational Databases, Query Optimization, Parallel and Distributed Databases, Data Integration, Large Scale, Data Grid Systems.

## 1 Introduction

At present, most of the relational database application programs are written in high-level languages integrating a relational language. The relational languages offer generally a declarative interface (or declarative language like SQL) to access the data stored in a database. Three steps are involved for query processing: decomposition, optimization and execution. The first step decomposes a relational query (a SQL query) using logical schema into an algebraic query. During this step syntactic, semantic and authorization are done. The second step is responsible for generating an efficient execution plan for the given SQL query from the considered search space. The third step consists in implementing the efficient execution plan (or operator tree) [51]. In this paper, we focus only on query optimization methods. We consider multi-join queries without "group" and "order by" clauses.

Work related to the relational query optimization goes back to the 70s, and began mainly with the publications of Wong et al. [138] and Selinger et al. [112]. These papers

A. Hameurlain et al. (Eds.): Trans. on Large-Scale Data- & Knowl.-Cent. Syst. I, LNCS 5740, pp. 211–242, 2009.

motivated a large part of the database scientific community to focus their efforts on this subject. The optimizer's role is to generate, for a given SQL query, an optimal (or close to the optimal) execution plan from the considered search space. The optimization goal is to minimize response time and maximize throughput while minimizing optimization costs.

The general problem of the query optimization can be expressed as follows [41]: let a query q, a space of the execution plans E, and a cost function cost (q) associated to the execution of $p \in E$, find the execution plan calculating q such as the cost (q) is minimum. An optimizer can be decomposed into three elements [41]: a *search space* [85] corresponding to the virtual set of all possible execution plans corresponding to a given query, a *search strategy* generating an optimal (or close to the optimal) execution plan, and a *cost model* allowing to annotate operators' trees in the considered search space.

Because of the importance, and the complexity of the query optimization problem [21, 75, 82, 103], the database community made a considerable effort to develop approaches, methods and techniques of query optimization for various Database Management Systems DBMS (i.e. relational, deductive, distributed, object, parallel) [7, 9, 21, 26, 47, 52, 61, 62, 79, 82, 103, 125]. The quality of query optimization methods depends strongly on the accuracy and the efficiency of cost models [1, 42, 43, 66, 99, 141].

There are two types of query optimization approaches [27]: static, and dynamic. During more than twenty years, most of the DBMSs have used the static optimization approach which consists of generating an optimal (or close to the optimal) execution plan, then executing it until the termination. All the methods, using this approach, suppose that the values of the parameters used (e.g. sizes of temporary relations, selectivity factors, availability of resources) to generate the execution plan are always valid during its execution. However, this hypothesis is often unwarranted. Indeed, the values of these parameters can become invalid during the execution due to several causes [98]:

1. *Estimation errors*: the estimation on the sizes of the temporary relations and the relational operator costs of an execution plan can be erroneous because of the absence, the obsolescence, and the inaccuracy of the statistics describing the data, or the errors on the hypotheses made by the cost model. For instance, the dependence or the independence between the attributes member of a selective clause (e.g. town='Paris' and country = 'France'). These estimation errors are propagated in the rest of the execution plan. Moreover, [70] showed that the propagation of these errors is exponential with the number of joins.
2. *Unavailability of resources:* at compile-time, the optimizer does not have any information about the system state when the query will run, in particular, about the availability of resources to allocate (e.g. available memory, CPU load).

Because of reasons quoted previously, the execution plans generated by a static optimizer can be sub-optimal. To correct this sub-optimality, some recent researches suggest improving the accuracy of parameter values used during the choice of the execution plan. A first solution consists in improving the quality of the statistics on the data by using the previous executions [1]. This solution was used by [20] to improve the estimation accuracy of the operator selectivity factors and by [117] to estimate the correlation between predicates. The second solution proposed by [80]

concentrates on the distributed queries. The optimizer generates an optimal (or close to the optimal) execution plan, having deduced the data transfer costs and the cardinalities of temporary relations. In this solution, the query operators are executed on a tuple subset of the operands to estimate the data transfer costs and the cardinalities of temporary relations. In both solutions, the selected execution plan is executed until the termination, whatever are the changes in execution environment.

As far as the dynamic optimization approach, it consists in modifying the sub-optimal execution plans at run-time. The main motivations to introduce 'dynamicity' into query optimization [27], particularly during the resource allocation process, are based on: (i) willing to use information concerning the availability of resources, (ii) the exploitation of the relative quasi-exactness of parameter values, and (iii) the relaxation of certain too drastic and not realistic hypotheses in a dynamic context (e.g. infinite memory). In this approach, several methods were proposed in different environments: uni-processor, distributed, parallel, and large scale [3, 4, 6, 7, 8, 12, 14, 15, 17, 18, 27, 30, 37, 48, 50, 56, 59, 72, 73, 74, 76, 87, 95, 98, 101, 102, 105, 106, 107, 140]. All these methods have the capacity of detecting the sub-optimality of execution plans and modifying these execution plans to improve their performances. They allow to the query optimization process to be more robust with respect to estimation errors and to changes in execution environment.

The rest of this paper is devoted to provide a state of the art concerning the evolution of query optimization methods in different environments (e.g. uni-processor, parallel, distributed, large scale). For each environment, we try to describe synthetically some methods, and to point out their main characteristics [67, 98], especially, the nature of decision-making (centralized or decentralized), the type of modification (re-optimization or re-scheduling), the level of modification (intra-operator and/or inter-operator), and the type of event (estimation errors, delay, user preferences).

The major contributions of this paper are: (i) understanding the mechanisms of query optimization methods with respect to considered environments and their constraints, (ii) pointing out their main characteristics which allow comparing them, and (iii) the reasons for which proposed methods become very sophisticated.

This paper is organized as follows: firstly, in section 2, we introduce two main search strategies (enumerative strategies, random strategies) for uni-processor relational query optimization. Then, in section 3 we present a synthesis of some methods in a parallel relational environment by distinguishing the two phase and one phase approaches. Section 4 provides global optimization methods of distributed queries. Section 5, describes, in data integration (mediation) systems, both types of dynamic optimization methods: centralized and decentralized. Section 6 is devoted to give an overview of query optimization in large scale environments, particularly in data grid environments. Lastly, before presenting our conclusion, in section 7, we provide a qualitative analysis of described optimization methods, and point out their main characteristics which allow comparing them.

## 2 Uni-processor Relational Query Optimization

In the uniprocessor relational systems, the query optimization process consists of two steps: (i) logical optimization which consists in applying the classic transformation

rules of the algebraic trees to reduce the manipulated data volume, and (ii) physical optimization which has roles of [90]: (a) determining an appropriate join method for each join operator by taking into account the size of the relations, the physical organization of the data, and access paths, and (b) generating the order in which the joins are performed [69, 84] with respect to a cost model. In this section, we focus on physical optimization methods. We begin at first, to define, characterize, and estimate the size of the search space. Then, we present search strategies. These are based either on enumerative approaches, or random approaches. Finally, we synthesize some analyses and comparisons stemming from performance evaluations of proposed strategies.

## 2.1 Search Space

### 2.1.1 Characteristics

In relational database systems [31, 120, 130], each query execution plan can be represented by a processing tree where the leaf nodes are the base relations and the internal nodes represent operations. Different tree shapes have been considered: left-deep tree, right-deep tree, and bushy tree. The Fig.1 illustrates tree structures of relational operators associated with the multi-join query R1∞ R2 ∞ R3 ∞ R4 ∞ R4.

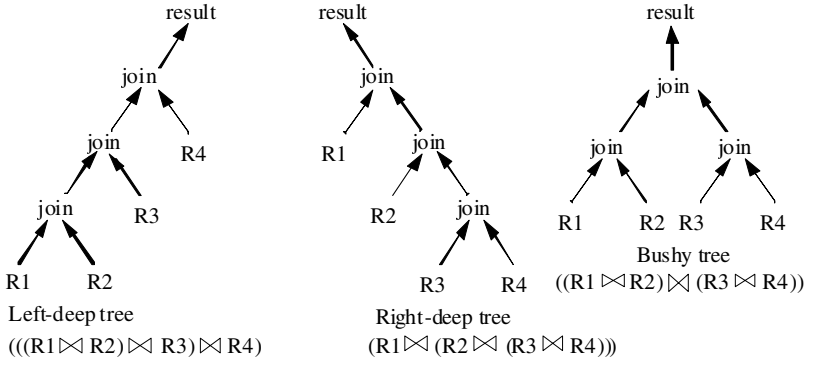


**Fig. 1.** Tree shape

A search space can be restricted according to the nature of the execution plans and the applied search strategy. The nature of execution plans is determined according to two criteria: the shape of the tree structures (i.e. left-deep tree, right-deep tree and bushy tree) and the consideration of plans with Cartesian products.

The queries with a large number of join predicates make the difficulty to manage associated search space which becomes too large. That is the reason why some authors [122, 123] chose to eliminate bushy trees. This reduced space is called valid space. This choice is due to the fact that this valid space represents a significant portion of the search space, which is the optimal solution. However, this assertion was never validated. Others, such as [100], think that these methods decrease the chances to obtain optimal solutions. Several examples [100] show the importance of the impact of this restrictive choice.

### 2.1.2 Search Space Size

The importance of the query shape[1] (i.e. linear, star or clique) and of the nature of the execution plans is due to their incidence on the size of the search space. If we have N relations in a multi-join query, the question is to know how many execution plans being able to be built, taking into account the nature of the search space. The size of this space also varies according to the shape of the query. In this case, [85, 124] proposed a table illustrating the lower and superior boundary markers of the search space by taking into account the nature of this one and characteristics of the queries which are: the type of a query (i.e. repetitive, ad-hoc), the query shape, and the size of a query (i.e. simple, medium, complex).

The results presented in [85, 124] point out the exponential growth of the number of execution plans according to the number of relations. This shows the difficulty to manage a solution set which is sometimes very large. Therefore, this brings the necessity of adapting the search strategy to the query characteristics.

## 2.2 Search Strategies

In the literature, we distinguish, generally, two classes of strategies allowing to solve the problem of the join scheduling for the query optimization:

- Enumerative strategies
- Random strategies.

The description of the principles of search strategies leans on the generic search algorithms described in [84] and on the comparative study between the random algorithms proposed by [69, 70, 86, 122, 123].

### 2.2.1 Enumerative Strategies

These strategies are based on the generative approach. They use the principle of dynamic programming (e.g. optimizer of System R). For a given query, the set of all possible execution plans is enumerated. This can lead to manage a search space too large in case of complex queries. They build execution plans from sub-plans already optimized by starting with all or part of base relations of a query. In the whole of generated solutions, only the optimal execution plan is returned for the execution. However, the exponential complexity of such strategies has led many authors to propose more efficient strategies. So enumerative strategies allow to discard bad states by introducing heuristics (e.g. depth- first search with different heuristics [123]). Several strategies are described in [84].

### 2.2.2 Random Strategies

The enumerative strategies are inadequate in optimizing complex queries because the number of execution plans quickly becomes too large [85, 124]. To resolve this problem, random strategies are used. The transformational approach characterizes this kind of strategies. Several rules of transformation (e.g.; Swap, 3Cycle, Join commutativity/ associativity) were proposed [69, 70, 122] where the validity depends on the nature of the considered search space [86].

---

[1] The query shape indicates the way where the relations are joined by means of predicates, as well as the number of referenced relations.

The random strategies start generally with an initial execution plan which is iteratively improved by the application of a set of transformation rules. The start plan(s) can be obtained through an enumerative strategy like Augmented Heuristics. Two optimization techniques were abundantly already studied and compared: the Iterative Improvement and the Simulated Annealing [68, 69, 70, 84, 122, 123].

The performance evaluation of these strategies is very hard because of strong influence, at the same time, of random parameters and factors. The main difficulty lies in the choice of these parameters (e.g local / global minimum detection, algorithm termination criterion, initial temperature, termination criterion of inner iteration). Indeed, the quality of execution and the optimization cost depend on the quality of choice. After the tuning of the parameters, the comparison of the algorithms will allow to determine the most efficient random algorithm for the optimization problem of complex queries. However, the results obtained by [122] and by [69] differ radically because, for [122], the Iterative Improvement algorithm is better than the Simulated Annealing, while for [69], we have the opposite (even if for these last ones, their conclusion remains more moderate). The parameters were determined thanks to experiments with various alternatives, in the case of [69], or by applying the methodology of the factorial experiments [122]. An example of the use of these factorial experiments is given in [121].

### 2.3 Discussion

In [69, 70, 122, 123], the authors concentrated their efforts on the performance evaluation of the random algorithms for Iterative Improvement and the Simulated Annealing. However, the difference of their results underlines the difficulty of such evaluation. Indeed, for Swami and Gupta [122, 123], the Simulated Annealing algorithm is never superior to the Iterative Improvement whatever the time dedicated to the optimization is, while for Ioannidis and Cha Kong [69, 70], it is better than the Iterative Improvement algorithm after some optimization time.

In [69, 70], the authors try to explain this difference. First, the considered search space is restricted to the left-deep trees in the case of Swami and Gupta [122, 123], while Ioannidis and Cha Kong [69, 70] study the search space in its totality. In [70], the authors spread their works on the study of the shape of the cost function by stressing the analysis of the linear and bushy spaces, and take into account only results waited in this restricted portion by the search space in order to keep the comparison coherent. The second difference concerns the join method. Swami and Gupta choose the hash join method, while Ioannidis and Kong [69, 70] use two join methods: nested loop and sort merge join. They choose even integrating the hash join method to show that their results do not depend on the method chosen. Another variant in the cost evaluation of the execution plan (CPU time for the first ones and I/O time for the second) has, either, no significant incidence on the difference of the results. On the other hand, they intuitively think that the number of nearest plans, the determination of the local minimum in the case of the Iterative Improvement algorithm and the definition of the transformation rules to be applied are important elements in the explanation of this difference. For example, if the number of nearest plans is not rather large, we can discard potential local minima and even indicate it as such, while they are not in reality. In that case, the results are skewed. The transformation rules applied by Swami

and Gupta [122, 123] generates nearest execution plans with a significant difference in cost [69]. Hence, the Simulated Annealing algorithm has no more the possibility of crossing a long moment in this zone of low-cost plans and offers then insufficient improvement. However, the algorithm of the Iterative Improvement can easily reach a local minimum.

The termination criterion of the Simulated Annealing defined in [122] does not give the time to the probability to decrease sufficiently. Indeed, when the time limit is reached, the probability to accept execution plans with high cost is still too high and the produced optimal plan has a still too expensive.

## 3 Parallel Relational Query Optimization

Parallel relational query optimization methods [57] can be seen as an extension of relational query optimization methods developed for the centralized systems, by integrating the parallelism dimension. Indeed, the generation of an optimal parallel execution plan (or close to optimal), is based on either a two-phase approach [60, 63], or on a one-phase approach [24, 86, 111, 142]. A two-phase approach consists in two sequential steps: (i) generation of an optimal sequential execution plan (i.e. logical optimization followed by a physical optimization), and (ii) resource allocation to this plan. The last step consists, at first, in extracting the various sources of parallelism, then, to assign the resources to the operations of the execution plan by trying to meet the allocation constraints (i.e. data locality, and various sources of parallelism). As far as the one-phase approach, the steps (i) and (ii) are packed into one integrated component [90]. The fundamental distinction between both approaches is based on the query characteristics and the shape of the search space [57].

In the proposals concerning parallel relational query optimization few authors [55, 61, 79] proposed a synthesis dedicated to parallel relational query optimization methods. Hasan et al [61] have briefly introduced what they consider the major issues to be addressed in parallel query optimization. The issues that are tackled in [79] include, mainly, the placement of data in the memory, concurrent access to data and some algorithms for parallel query processing. These algorithms are restricted to parallel joins. As far as proposals [55], the authors describe, in a very synthetic way, data placement, static and dynamic query optimization methods, and accuracy of the cost model. Nevertheless, the authors do not show how we can compare the two optimization approaches, and how we can choose the appropriate optimization approach. Last year, Taniar et al. [126] provide the latest principles, methods and techniques of parallel query processing in their book.

The rest of the section is devoted to provide an overview of some static and dynamic query optimization methods in a parallel relational environment by distinguishing the two phase and one phase approaches [57].

### 3.1 Static Parallel Query Optimization Methods

In this sub-section, we describe some one-phase and two-phase optimization strategies of parallel queries in a static context.

### 3.1.1 One-Phase Optimization

In a one-phase approach, Schneider et al. [111] propose a parallel algorithm to process a query compound of N joins for each search space shape (i.e. left-deep tree, right-deep tree and bushy tree, Cf. Fig. 1). The authors consider two methods of hash join: the simple hash join and the hybrid hash join. [111] reports for each search space shape, the need in memory size, the potential scheduling, and the capacity to exploit the different forms of parallelism. The study includes the case where the memory resource is unlimited and the more realistic case where the memory is limited. In the first case, the right deep tree is the most adapted to best exploit the parallelism. But, this structure is no longer the best when the memory is limited. Indeed, there are several strategies allowing to exploit the capabilities of the right deep trees when the memory is limited. The strategy, named "Static Right Deep Scheduling" [111], consists in cutting the right deep tree in several separate sub-trees in a way that the sum of the sizes of all the hash tables of a sub-tree can fit in memory. The temporary results of the execution of sub-trees T1, T2 …Tn will be stored in disks. The drawback of this strategy is that the number of sub-trees increases with the number of base relations which are not held stored in memory. Hence, this method reduces the pipeline chain and increases the response time. Two methods were proposed, one is based on segmented right-deep trees [24], and the other one is based on zigzag trees [142]. The objective of these two methods is to avoid the investigation of the bushy tree search space and then simplifying the optimization process.

### 3.1.2 Two-Phase Optimization

In the two-phase approach, Hasan et al. [23, 60] propose several scheduling strategies of pipelined operators. To improve the response time, they develop an execution model ensuring the best trade-off between parallel execution and communication overhead. Several scheduling algorithms (i.e. processor allocation) are then proposed. They are inspired by the heuristic LPT (Largest Processing Time). These algorithms exploit pipeline and intra-operation (partitioned) parallelisms. Indeed, the authors firstly propose scheduling algorithms exploiting only the pipeline parallelism (POT Pipelined Operator Tree Scheduling), then they show how to extend these algorithms to take into account the intra-operation parallelism and the communication costs. The scheduling principle of the POT is decomposed into several steps [23]: (i) generation of operators' monotonous tree [60] from operators' tree, (ii) fragmentation of the monotonous tree which consists in cutting the monotonous tree in a set of fragments, and (iii) scheduling which consists in assigning processors to fragments. The main difficulty lies in the determination of the number of fragments and the size of each fragment by insuring the best tradeoff between parallel execution - communication overhead.

As for the works of Garofalakis et al. [44, 45], they can be seen as an elegant extension of the propositions of [23, 41, 60]. Indeed, the works of [44, 45] take into account the fact that the parallel query execution requires the allocation of several resource types. They also introduce an original way to resolve this resource allocation by a simultaneous scheduling (e.g. parallelism extraction) and mapping method. First, [44] present a scheduling and mapping static strategy on a shared nothing parallel architecture, considering the allocation of several “preemptive” resources (e.g. processors). Next, the authors extend their own works in [45] for hybrid multi-processor architecture. This

extension consists, mainly, in taking into account the "no preemptive" resource (e.g. memory) in their scheduling and mapping method.

### 3.2 Dynamic Parallel Query Optimization Methods

The main motivations to introduce 'dynamicity' into query optimization [27], in particular in the resource allocation strategies, are based on: (i) the will to use,, information concerning the availability of the resources to allocate, (ii) the exploitation of the relative quasi-exactness of the metrics, and (iii) the relaxation of certain too drastic and not realistic hypotheses in a dynamic context. This sub-section describes in a synthetic way some one-phase and two phase parallel query optimization strategies. It should be pointed out that the proposed resource allocation methods become very complex and sophisticated in such a dynamic context.

#### 3.2.1 One-Phase Optimization

In this approach, the majority of work point out the importance of the determination of the join operation parallelism degree and the resource allocation method (e.g. processors and memory). Thus, it becomes interesting to synthesize some methods proposed in the literature, mainly [19, 81, 96, 107].

In their most recent work Brunie et al. [18, 19, 81] are not only interested in a multi-join process in a multi-user context, but also consider the current system state in terms of multi-resource contention. [18] studied, more generally, the relational query optimization on shared nothing architecture. The optimizer MPO (Modular Parallel query Optimizers) [81] determines dynamically the intra-operation parallelism degree of the join operators of a bushy tree. The authors suggest a dynamic heuristic to resource allocation in four steps applied in the following order: (i) Preservation of the data locality (or "data localization"), (ii) Size of the memory, (iii) I/O Reduction, and (iv) Operation serialization of a bushy tree:

The proposals of Mehta et al. [96] and Rahm et al. [107] were developed independently of one-phase and two-phase approaches. Furthermore, their proposals are very representative and describe relevant and original solutions with respect to the problems identified above (i.e. determination of the parallelism degree and the resource allocation methods), we chose to include them in the one-phase approach.

Mehta et al. [96] propose four algorithms (Maximum, MinDp, MaxDp, and RateMatch) to determine the join parallelism degree independently of the initial data placement. The originality of the algorithm Rate tries to make correspond the production rate of the result tuples of an operator with the consumption rate of next operator tuples. Then, the authors describe six alternative methods of processor allocation in the clones of a unique join operator. They are based on heuristics such as the random or round-robin strategies, and on a model taking into account the effect of the resource contention.

As for the proposals of Rahm et al. [107], who extend the works of [95], they tackle the problem of the dynamic workload balancing of several queries compounded in a single hash join on a shared nothing architecture. The intra-operation parallelism of a join as well as the choice of the execution processors of the join are determined in a "integrated" way (.i.e. in a single step) by considering the current system state. This state is characterized by using the resources "bottlenecks": CPU, memory, and disk.

### 3.2.2 Two-Phase Optimization

*XPRS adapting scheduling method*

In the system XPRS (eXtended Postgres one Raid and Sprite) [118], implanted on shared memory parallel architecture, Hong [63] proposes an adaptive scheduling method of fragments stemming from the best sequential execution plan represented by a bushy tree. Fragments are used as unity of parallel execution and they will also be called *tasks* in this sub-section. The adaptive scheduling algorithm is based on the following three elements: (i) classification of the "IO-bound" and "CPU-bound" tasks, (ii) computing method of the IO-CPU balance point of two tasks, and (iii) mechanism of dynamic adaptation of the parallelism degree of a task.

The proposed strategy by [63] consists in finding task scheduling which maximizes the use of the resources (i.e. processors and disks), and thus minimizes the response time. For that purpose, [63] defines two types of tasks: the IO-bound tasks (limited by Input / Output) and the CPU-bound tasks (limited by the number of processors). To maximize the resource utilization (e.g. when one of both tasks ends, a part of resources remains unused), [63] proposes a dynamic adaptation method of the parallelism degree of a task according to the implemented distribution methods (i.e. round-robin, interval). This method is used in the adaptive scheduling so that the system always works on the IO-CPU balance point.

*Dynamic re-optimization methods of sub-optimal execution plans*

In Kabra et al. [77], where the idea is close Brunie and al. [18], the authors propose a dynamic re-optimization algorithm which detects and corrects sub-optimality of the execution plan produced by the optimizer at compile time. This algorithm is implanted in the system Paradise [33] which is based on the static optimizer OPT++ [78]. The authors show that sub-optimality of an execution plan can result: (i) in a poor join scheduling, (ii) in the inappropriate choice of the join algorithms, or (iii) in a poor resources allocation (CPU and memory). These three problems would be caused by erroneous or obsolete cost estimations, or another lack of information necessary for the static optimization, concerning to the system state. The basic idea of this algorithm is founded on the collection of the statistics in some key-points during the query execution. The collected statistics correspond to the real values (observed during the execution), where the estimation is subject to error at compile time (e.g. size of a temporary relation). These statistics are used to improve the resource allocation or by changing the execution plan of the remainder of the query (i.e. the part of the query, which is not executed yet).

As for the re-optimization process, it will be engaged only in case of estimation errors really bringing sub-optimality besides of the execution plan. Indeed, on the basis of these new improved estimations, if they are different in a significant way from those supplied by the static optimizer a new execution plan of the remainder of the query is generated in the case where it brings a minimum benefit.

## 4 Distributed Query Optimization

The main motivation of the distributed databases is to present data which are distributed on networks of type LAN (Local Area Network) or of type WAN (Wide Area

Network) in an integrated way to a user. One of the objectives is to make data distribution transparent to the user. In this environment, the main steps of the evaluation process of a distributed query are data localization and optimization. The optimization process [82, 103] takes into account network particularities. Indeed, contrary to the interconnection network of a multi-processor, networks have a lower bandwidth and a more important latency. For example, with a satellite connection the latency exceeds the half-second. These particularities are significant in cost of a distributed execution plan that authors [10, 103] are focused. They suppose that the communication cost is widely superior to those of the I/O and the CPU. So, many works focus on the communication cost to the detriment of CPU and I/O costs. At present, with the improvement of network performance, the cost functions used by the optimization process take into account the processing (i.e. CPU and I/O) and communication time together.

The optimization process of a distributed query is composed of two steps [103]: a global optimization step and a local optimization step. The global optimization consists of: (i) determining the best execution site for each local sub-query considering data replication, (ii) finding the best inter-site operator scheduling, and (iii) placing these last ones. As for local optimization, it optimizes the local sub-queries on each site which are involved to the query evaluation. The inter-site operator scheduling and their placement are very important in a distributed environment because they allow to reduce the data volumes exchanged on the network and consequently to reduce the communication costs. Hence, the estimation accuracy of the temporary relation sizes that must be transferred from a site to another one is important. In the rest of this section, we present global optimization methods of distributed queries. They differ by the objective function used by the optimization process and by the type of approach: static or dynamic.

### 4.1 Static Distributed Query Optimization

In distributed environments, various research works concerning the static query optimization are focused mainly on the optimization of inter-site communication costs. The idea is to minimize the data volume transferred between sites. In this perspective, there are two methods to process inter-site joins [103]: (i) the direct join by moving one relation or both relations, and (ii) the join based on semi-join. This alternative consists in replacing a join, whatever the class of algorithm implanting this join is, by the combination of a projection, and a semi-join ended by a join [25]. The cost of the projection can be minimized by encoding the result [133]. The benefit of a join based on semi-join with respect to a direct joint is proportional in the join operator selectivity [134]. According to the relation profiles (e.g. relation size), the optimizer will choose the approach which minimizes the data volume transferred between sites. For example, the SDD-1 system [10] often uses the join based on semi-join. However, System R* [113] avoids to use it. Indeed, the use of a join based on semi-join can increase the query processing time. Mackert and Lohman [91] showed the importance of the local processing cost in the performance of a distributed query. Furthermore, its consideration by the optimizer significantly increases the size of the search space. Indeed, in a query, there are several possibilities of join based on semi-join for a given relation. The number of join based on semi-join is an exponential function

which depends of the number of temporary relations resulting from local sub-queries [103]. This explains why many optimizers do not use this alternative.

The quality of a distributed execution plan which is generated by the global optimization process depends on the accuracy of the used estimations. However, it is difficult to estimate the parameters (e.g. relation profile, resource availability) used by the optimizer. Generally, the used cost models made the assumption of processor and network uniformity. These cost models assume that all processors and network connections have the same speed and bandwidth, like in a parallel environment. Furthermore, they do not take into account the workload of processors nor that of the network. Based on these observations, several works [80, 119] try to improve the accuracy of these parameters. In this objective the Mariposa distributed DBMS [119] leans on an economic model in which querying servers buy data from data server. Each query Q, which is decomposed into several sub-queries Q1, Q2,…, QN, is administered by a broker. A broker obtains bids for a sub-query Qi from various sites. After choosing the better bid, the broker notifies the winning site. The advantage of this method is that it leans on the local cost models of every DBMS which can participate in the query evaluation. So, it considers the processor heterogeneity and takes into account their workload.

[80] propose that the optimizer generates an optimal (or close to the optimal) execution plan, having deduced the data transfer costs and the cardinalities of temporary relations. In this solution, the operators of a query are executed on a tuple subset of the operands to estimate the data transfer costs and the cardinalities of temporary relations. After deduced the cost of these parameters, an optimal execution plan is generated and executed until the termination, whatever the changes in execution environment are.

### 4.2 Dynamic Distributed Query Optimization

A solution to correct the sub-optimality of an execution plan consists in changing the operation scheduling at run-time. In the multi-database MIND system, Ozcan et al. [102] proposed strategies for dynamic re-scheduling of inter-site operators (e.g., join, union) to react to the inaccuracies of estimations. The inter-site operators can be executed as soon as two sub-queries which are executed on different sites produced their results. These strategies use the partial results available at run-time to define the scheduling of the executions between the inter-site operators. The query processing is done in two steps [37]:

1. Compilation. During this step, a global query is decomposed into local sub-queries. The sub-queries are sent to different sites to be executed in parallel.
2. Dynamic scheduling. This step defines a dynamic scheduling between the operations consuming the results of sub-queries sent on sites. When a sub-query produces its result, a threshold is associated to the result. This threshold is used to determine if the result must be consumed immediately to execute a join with another result already available, or if the consumption of this result will be delayed while waiting for another result, which is unavailable in this moment. The threshold associated with a result is calculated according to the costs and selectivity factors of all joins connected to this result.

This scheduling strategy reduces the uncertainty of estimations since it is based on the execution times of local sub-queries. Moreover, it avoids the needs to know the cost models of the various databases.

## 5 Query Optimization in Data Integration Systems

Data integration systems extend [22, 53, 88, 127, 128, 136] the distributed database approach to multiple, autonomous, and heterogeneous data sources by providing uniform access (same query interface in read only to all sources). We use the term data source to refer any data collection which his owner wishes to share with other users. The main differences of a distributed database approach are the number of data sources and the heterogeneity of the data sources. The distributed database approach addresses about tens of distributed databases while data integration system approach can scale up to hundreds of data sources [104]. In addition to the material heterogeneity (i.e. CPU, I/O, network) due to the environment, the data sources are heterogeneous by their data structure (e.g. relational or object). Moreover, the software infrastructures allowing the access to data sources have different capabilities for processing queries. For example, a phone book service which requires the name of a person to return a phone number is a data source where the access is restricted. In this context, we need new operators in order to access to data sources and to, for instance, join two relations. Consider an execution plan that needs a relational join between Employee (empId, name) and Phone (name, phoneNumber) tables on their name attribute. In a standard join both of the following fragments: Join (Employee, Phone) and Join (Phone, Employee) are valid since join is a commutative operator. However, with restricted sources, the second fragment Join (Phone, Employee) on name attribute is not valid, since Phone requires the value of the name attribute in order to return the value of the phoneNumber. In consequence, we need a new join operator which is asymmetric in nature, also known as dependent join Djoin [46]. The asymmetry of this operator causes the search space to be restricted and raises the issue of capturing valid (feasible) execution plans [92, 93, 139].

In an environment with hundreds of data sources connected on Internet it is even more difficult to estimate, at compile time, the availability of the resources like network, CPU or memory. Hence, many authors propose dynamic optimization strategies to correct the sub-optimality of execution plans at runtime. Initially, proposed methods are centralized [3, 4, 7, 14, 15, 32, 74, 109]. A dynamic optimization method is said to be centralised if there is a unique process, generally the optimiser, which is charged to supervise, control and modify the execution plans. This process can be based on other modules ensuring the production of necessary information for the modifications and the control of an execution plan. On other hand, in this environment, two phenomena that occur frequently are significant: initial delays before data start arriving and bursty arrivals data thereafter [72]. In order to react to these unpredictable data arrival rate, several authors propose to decentralize the control inside the operator [72, 131, 132]. The idea is to produce most quickly as possible a part of the result with the already arrived tuples during the waiting of operand tuples.

In the rest of the section, we present the specific operators to the data integration, at first, then we describe both types of dynamic optimization methods: centralized and decentralized.

### 5.1 Operators for Restricted Source Access

Consider the execution plan presented previously that needs a relational join between Employee (empId, name) and Phone (name, phoneNumber) tables. The tables can be modeled with the concept of 'binding patterns' as introduced in [108]. Binding patterns can be attached to the relational table to describe its access restrictions due to the reasons of confidentiality or performance issues. A binding pattern for a table R(X1, X2, . . . , Xn) is a partial mapping from {X1, X2, . . . , Xn} to the alphabet {b, f} [93]. 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 relational scan operator). The binding patterns of the tables of our example are as follows: Employee ($empId^f$, $name^f$), and Phone ($name^b$, $phoneNumber^f$). It means that the Employee table is ready to return the values of the empId, and the name while the Phone table can give the phoneNumber only if the value of the name attribute is known. Regular set of relational operators are insufficient in order to answer queries in the presence of restricted sources.

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 relational scan and relational join. In the example, in order to get the phoneNumber we have to give the values of the name attribute. So we need a new scan operator which is able to deal with the restricted sources. We quote this operator 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, this new operator DAccess returns different sets depending on its input set. Formal semantics of DAccess is as follows: Consider a table $R(X^b, Y^f)$ and $\chi$ be a set of values for X. Then, $DAccess(R(X^b, Y^f))\chi = \sigma\, X \in \chi(R(X, Y))$ [93].

We noticed that to make the join between Employee ($empId^f$, $name^f$), and Phone ($name^b$, $phoneNumber^f$) we need a new join operator known as dependent join [46], represented by the symbol $\overrightarrow{\bowtie}$. The representation of the dependent join is $T \leftarrow Scan(R1(U^f, V^f)) \overrightarrow{\bowtie}_{V=X} DAccess(R2(X^b, Y^f))$. The hash dependent join consists in building a hash table from R1 and at the same time the distinct values of the attribute(s) V are retrieved and stored them into a table P. P is given to the DAccess operator to compute $R2' = \sigma\, X \in P\ (R2(X, Y))$. Then the hash table is probed with R2' to compute the result.

### 5.2 Centralized Dynamic Optimization Methods in Data Integration Systems

In this sub-section, we present some dynamic optimization methods and techniques where the type of decision-making is centralized. We classify these methods according

to the modification level of execution plans. This modification can be taken either on the intra-operator level, or on the inter-operator level.

### 5.2.1 Modification of Execution Plans on the Intra-operator Level

The sub-optimality of execution plans can be modified during the execution of an operator (intra-operator). With this objective, two approaches were proposed: the first one is based on the routing of tuples named Eddy [7], and the second one is based on the dynamic partitioning of data [74].

Avnur and Hellerstein [7] proposed a mechanism named Eddy for query processing which updates continuously the execution schedule of operators in order to adapt to the changes in execution environment. Eddy can be considered as a router of tuples positioned between a number of data sources and a set of operators. Each operator must have one or two input queues to receive the tuples sent by Eddy and an output queue to return the result tuples to Eddy. The tuples received by an Eddy are redirected towards the operators in different orders. Thus, the scheduling of operators is encapsulated by the dynamic routing of tuples.

The key point in Eddy is the routing of tuples. Thus, the policy of the tuple routing must be efficient and intelligent in order to minimize the query response time. For that purpose, several authors [32, 109] suggest to extend Eddy's mechanism to improve the quality of the routing.

Dynamic data partitioning was proposed by Ives et al. [74]. It corrects the sub-optimality of execution plans relying on dynamic data partitioning. In this method, a set of execution plans is associated to each query which will be executed either in parallel or in sequence on separate data partitions. The execution plan of a query is constantly supervised at runtime, and it can be replaced by a new plan in the case where the current plan is considered to be sub-optimal. The tuples which are processed by each used plan represent a data partitioning. When an execution plan is replaced, a new data partitioning is produced. Each used execution plan produces a part of the total result from the associated data partitioning during the query execution. The union of the tuples produced by the various used execution plans provides only part of the total result. Thus, to calculate the final result of the query, it must also calculate the results of all the combinations of various data partitioning.

This method is similar to that of Eddy [7]. But contrary to Eddy which uses a local decision routing, this method is based on more total information to generate the new plans. The main difference is that the decision to suspend or replace an execution plan by another one is made by the optimizer.

### 5.2.2 Modification of Execution Plans on the Inter-operator Level

A solution to correct the sub-optimality of execution plans consists in changing the operation scheduling at runtime. The works of Amsaleg et al. [3] take into account the delays in data arrival rates. They have identified three types of delays: (i) Initial delay: that occurs before the arrival of the first tuple, (ii) bursty arrival: the data arrive in bursts but the arrival of these data is suddenly stopped and followed by a long period of no arrival, and (iii) slow delivery: the data arrive regularly but slower than normal. To deal with these delays, two methods were proposed by Amsaleg et al. [4] and by Bouganim et al. [14, 15].

The technique of query scrambling [3, 4] was proposed to process the blockings caused by the delays in data arrival rates. It tries to mask these delays by the executions of other portions of the execution plan until the termination of these delays. The technique of query scrambling processes the initial delay and the bursty arrival in two phases [3]:

1. Re-scheduling: as soon as a delay is detected, this phase is invoked. It begins with the separation of the relational operators of an execution plan in two disjoined sets: (i) the set of blocked operators that contains all the ancestors of unavailable operands, and (ii) the set of executable operators that contains the remainder of the operators that do not belong to the set of blocked operators. Then, a maximum executable sub-tree is extracted from the set of the executable operators. This maximum sub-tree is executed and its intermediate result is materialized.
2. Synthesis: this phase is invoked if the set of the executable operators is empty and the set of the blocked operators is not empty. Contrary to the re-scheduling phase, the synthesis phase can significantly change the execution plan by adding new operators and/or by removing existing operators. The synthesis phase starts, at first, by the construction of a graph of the joins which are ready to be executed. Then, a join is processed and the result is materialized. The synthesis phase is finished if all delays are finished, or if the graph is reduced to only one node or several nodes without join predicates.

The technique of query scrambling supposes that an execution plan is executed without taking into account the delays in data arrival rates during plan execution. For that, Bouganim et al. [14, 15] proposed a strategy where the memory is available and data arrival rates are constantly supervised. This information is used to produce a new scheduling between the various fragments of the execution plan or to re-optimize the remainder of the query.

The paper of Ives et al. [72] described a dynamic optimization method which is able to deal with the majority of the changes in execution environment (delays, errors and unavailable memory). This method interweaves the phases of optimization and execution and it uses specific dynamic operators. In this method, the optimizer transforms a query into an annotated execution plan [77] and generates the associated rules with Event-Condition-Action type. These rules determine the behavior of the execution plan according to the changes at runtime. They check certain conditions (e.g. comparison of the sizes of the current temporary relations with those estimated during compilation) when events occur (e.g. delay, memory unavailable) they start actions (e.g. memory re-allocation, re-scheduling or re-optimization).

### 5.3 Decentralized Dynamic Optimization Methods in Data Integration Systems

The decentralized dynamic optimization methods correct the sub-optimality of execution plans by decentralizing the control. The conventional hash join [16] algorithm requires the reception of all tuples of the first operand for building the hash table before beginning the probe step. Thus, the time to produce the first tuple can be long if: (i) the size of the operands is large, or (ii) when the data arrival rate is irregular. Contrary to the conventional hash join, the double hash join (DHJ) introduced by Ives et al. [72] built a hash table for each operand. When a tuple arrives, it is inserted firstly in the associated hash table. Then, it is used to probe the other hash table. If the probe step allows

to produce result tuples, then these tuples are immediately delivered. DHJ was proposed in TUKWILA project [72] to deal with the problems of conventional hash join in the context of data integration: (i) the production time of the first tuple is minimized, (ii) the optimizer does not need to know the sizes of the operands in order to choose the operand used in the building of the hash table, and (iii) it masks the slow arrival rate of tuples from an operand by processing the tuples of the other operand.

However, DHJ requires to maintain the two hash tables in memory. This can limit the use of DHJ with operands having large sizes or with queries constituted of several joins. To solve this problem, parts of the hash tables residing in the memory are moved towards a secondary storage space. When the memory becomes saturated, a partition of one of the two tables is chosen to be moved towards the secondary storage space.

The DHJ allows reducing the necessary time for the production of the first tuple of result. Moreover, it makes it possible to continue the production of the result tuples in spite of the unavailability of any one of the two operands. However, it can lead to bad performances if the tuple productions of the two operands are blocked. For that, the Xjoin operator is proposed by Urhan et Franklin [131]. When Xjoin detect the unavailability of the tuples of each operand, the tuples of a portion resident in the secondary storage space are joined with the tuples of the same partition of second operand residing in memory.

To accelerate the production of result tuples, it is interesting to define scheduling mechanisms between the various phases of the Xjoin operator. For that purpose, Urhan and Franklin [132] proposed a scheduling technique using the notion of Stream. Stream is the execution unit which consumes and produces tuples. The execution schedule of Stream is determined at runtime and is changed according to the variations of the system behaviour (productions of tuples, terminated streams).

# 6 Query Optimization in Large Scale Environments

## 6.1 Query Optimization in Large Scale Data Integration Systems

Large scale environment means [58]: (i) high numbers of data sources (e.g. databases, xml files), users, and computing resources (i.e. CPU, memory, network and I/O bandwidth) which are heterogeneous and autonomous, (ii) the network bandwidth presents, in average, a low bandwidth and strong latency, and (iii) huge volumes of data.

In a large scale distributed environment, performances of previous optimization methods decrease because: (i) the number of messages relatively important on a network with low bandwidth and strong latency, and (ii) the bottleneck that forms the optimizer. It becomes thus convenient to make the query execution autonomous and self-adaptable. In this perspective, two close approaches have been investigated: the broker approach [28], and the mobile agent approach [6, 76, 101, 110]. The second approach consists in using a programming model based on mobile agents [40], knowing that at present the mobile agent platforms supply only migration mechanisms, but they do not offer proactive migration decision policy.

The rest of this sub-section is devoted to describe execution models associated to brokers and mobile agent approaches [6, 28, 66, 76, 98, 101, 110].

*Broker Approach*
In a large scale mediation system context, Collet and Vu [28] proposed an execution model based on brokers. The broker, which is the basic unit of the query execution, supervises the execution of a sub-query. It detects the estimation inaccuracies and adapts itself according to these inaccuracies. Moreover, it communicates with the other brokers to take into account the updates of the execution environment. The principal components of a broker are: (i) context including the annotations and constraints necessary for the execution of a sub-query, (ii) operator of the sub-query, (iii) buffer allowing to synchronize the data exchange between the brokers, and (iv) rules which define behavior of the broker according to changes of the execution environment.

*Mobile Agent Approaches*
A mobile agent [40] is an autonomous software entity which can move (code, data, and execution state) from a site to another in order to carrying out a task. In the traditional operating system, the decision of migration activity is controlled by another process. However, in a mobile agent, the decision of the migration activity is made by the agent itself.

The operators of double hash join and Xjoin improve the local processing cost by adapting the use of resources CPU, I/O and memory with the changes of the execution environment (e.g. estimation errors, delays in data arrivals rates) and does not take in account the network resource. In objective to take into account the network resource, the work proposed by Arcangeli et al. [6], Hussein et al. [66] and Ozakar et al. [101] based on mobile agents extend the algorithms of direct join, semi-join based join and dependent join (in presence of binding patterns). This extension allows them to change their execution sites proactively. Each mobile agent executing a join chooses itself its execution site by adapting to the execution environment (e.g. CPU load, bandwidth) and the estimation accuracies on temporary relation sizes. Hence, the control which makes the decision of the execution site change is carried out in a decentralized and autonomous way. Furthermore, for dynamic query optimization, Morvan et al. [97] proposed three cooperation methods between the mobile join agents. These methods allow to a mobile agent to make its decision to migrate or not according to the decisions of the other agents communicating with it. These methods minimize the number of messages exchanged between agents.

As far as work of Jones and Brown [76], they propose, for large scale distributed queries, an execution model based on mobile agents which react to the estimations inaccuracies. The mobile agents are charged to execute the local sub-queries of an execution plan. These agents compare the partial results (e.g. size, execution costs) with the estimations used during compilation in order to detect sub-optimality. By taking into account the possibility of migration of mobile agents, two strategies were proposed:

1. *Decentralized execution without migration*: the agents, executing sub-queries, communicate between them, by broadcasting their partial execution states, in order to produce an execution plan for the remainder of the query.
2. *Decentralized Execution with migration*: this strategy extends the previous strategy while allowing the agents to migrate from one site to another before beginning their executions. The decision of migration can be made in a distributed, individual or centralized way.

Another method based on mobile agents has been proposed by [110] in order to execute queries in a web context. In this context, the query result can correspond to a new query on another server which processes it. For this, two mechanisms were proposed which are also known as being parts of LDAP (Lightweith Directory Access Protocol) [64]: (i) referral which consists into return to the user, the new query and server address to process it, and (ii) chaining which consists in cooperating with the server executing the new query to produce the result.

In this approach [110], the mobile agents are used to exploit these two mechanisms in the query processing. Each query is processed by using a mobile agent which can choose the best adapted mechanism (referral and chaining).

### 6.2 Query Optimization in Data Grid Systems

Since more than ten years, the grid systems are very active research topics. The main objective of grid computing [39] is to provide a powerful and platform which supplies resources (i.e. computational resources, services, metadata and data sources). The grid computing is very important for scale distributed systems and applications that require effective management distributed and heterogeneous resources [58]. Large scale and dynamicity of nodes (unstable system) characterize the grid systems. Dynamicity of nodes (system instability) means that a node can join, leave or fail at any time. Today, the grid computing, intended initially for the intensive computing, open towards the management of voluminous, heterogeneous, and distributed data on a large-scale environment. Grid data management [104] raises new problems and presents real challenges such as resource discovery and selection, query processing and optimization, autonomic management, security, and benchmarking. To tackle these fundamental problems [104], several methods have been proposed [5, 30, 48, 49, 65, 94, 129]. A very good and complete overview addressing the most above fundamental problems is described in [104]. The authors discuss a set of open problems and new issues related to Grid data management using, mainly, Peer-to-Peer P2P techniques [104]. More focused on a specific and very hot problem such as resource discovery, [129] propose a complete review of the most promising Grid systems that include P2P resource discovery methods by considering the three main classes of P2P systems: unstructured, structured, and hybrid (super-peer). The advantages and weaknesses of a part of proposed methods are described in [104, 129].

The rest of this sub-section tries to provide an overview of query processing and optimization in data grid systems.

Several approaches have been proposed for distributed query processing (DQP) in data grid environments [2, 5, 48, 49, 50, 65, 115, 135]. Smith et al. [115] tackle the role of DQP within the Grid and determine the impact of using Grid for each step of DQP (e.g. resource selection). The properties of grid systems such as flexibility and power make grid systems suitable platforms for DQP [115].

In recent years, convergence between grid technologies and web services leads researchers to develop standardized grid interfaces. Open Grid Services Architecture OGSA [38] is one of the most well known standards used in grids. Many applications are developed by using OGSA standards [2, 5, 135]. OGSA-DQP [2] is a high level data integration tool for service-based Grids. It is built on a Grid middleware named OGSA-DAI [5] which provides a middleware that assists its users by accessing and

integrating data from separate sources via the Grid. [135] describes the concepts that provide virtual data sources on the Grid and that implement a Grid data mediation service which is integrated into OGSA-DAI.

By analyzing the approaches of DQP on the Grid, the research community focused on the current adaptive query processing approaches [7, 47, 62, 67, 74] and proposed extensions in grid environments [29, 48, 50]. These studies achieve query optimization, by providing efficient resource utilization, without considering parallelization. Although, they use different techniques, most of the studies profit existing monitoring systems to determine progress of the queries. In [48], Gounaris et al. highlighted the importance and challenges of DQP in Grids. They mentioned the necessity of grids by emphasizing increasing demand for computation in the distributed databases. They also explained the challenges in developing adaptive query processing systems by expressing the weaknesses of existing studies and key points for the solutions. After giving the challenges, Gounaris et al. [50] proposed an adaptive query processing algorithm. They introduced an algorithm which provides both a resource discovery/allocation mechanism and a dynamic query processing service. In [114], Slimani et al. developed a cost model by modeling the network characteristics and heterogeneity. By using this cost model, they also introduced a query optimization method on top of Beowulf clusters [34]. They considered both logical and physical costs and deployed the distributed query according to the cheapest cost model. In [29], Cybula et al. introduced a different technique for query optimization which is based on caching of query results. They developed a query optimizer which stores results of queries inside the middleware and used the cache registry to identify queries that need not be reevaluated.

As far as parallelism dimension integration, many authors have re-studied DQP in order to be efficiently adopted by considering the properties (e.g. heterogeneity) of grids. Several methods are proposed in this direction [13, 30, 49, 89, 106, 116] which define different algorithms for parallel query processing in grid environments. The proposed methods consider different forms of parallelism (e.g. pipelined parallelism), whereas all of them consider also resource discovery and load balancing. In [13], Bose et al. examined the problem of efficient resource allocation for query sub-plans. They developed their algorithm by exploiting the bushy query trees. They incrementally distributed the sub-queries until a stopping condition is satisfied. In [30, 106] the authors introduced an adaptive parallel query processing middleware for the Grid. They developed a distributed query optimization strategy which is then integrated with a grid node scheduling algorithm by considering runtime statistics of the grid nodes. Gounaris et al. [49] proposed an algorithm which optimizes parallel query processing in grids by iteratively increasing the number of nodes which execute the parallelizable sub-plans. In [89], Liu et al. presented a query optimization algorithm which grades the nodes according to their capacities. They determined serial and parallel parts of the queries and proposed an execution sequence in highest ranked nodes. Soe et al. [116] proposed a parallel query optimization algorithm. In their study, they considered resource allocation, intra-query parallelism and inter-query parallelism by analyzing bushy query trees.

## 7 Discussion

According to the discussion led in the section 2.4, and the results in [122, 123, 68, 69, 70, 84], it is difficult to conclude about the superiority of a search strategy (e.g. scheduling of the join operators) with regard to the one another . However, each of them proposes a solution to improve the performances of these algorithms. Ioannidis and Kong [69, 70] chose to propose a new algorithm, called Two Phase Optimization [69], which consists in applying, at first, the Iterative Improvement algorithm, and then, the Simulated Annealing algorithm. As for Swami [123], he chose to experiment a set of heuristics with the aim of improving the performances of the Iterative Improvement and the Simulated Annealing algorithms [123]. The works of Ioannidis and Kong was able to show that the choice of a join method has no direct influence on the performances of the search strategies. In a parallel environment, Lanzelotte and al. [86] showed that the search strategy in breath first is not applicable in a bushy search space for queries with 9 relations or more. The use of a random algorithm is then indispensable. The authors thus developed a random algorithm called Toured Simulated Annealing in a context of parallel processing [86].

The search strategies find the optimal solution more or less quickly according to their capacity to face the various problems. They must be adaptable to queries of diverse sizes (simple, medium, complex) and in various types of use (i.e. ad-hoc or repetitive) [54, 83]. A solution to this problem is the parameterization and the extensibility of query optimizers [71, 83] possessing several search strategies, each being adapted for a type of queries. The major contributions in this domain arise, mainly, from the Rodin project [83, 84, 85, 86] as well as on the Ioannidis and Kong's results [69]. Indeed, one of the main aspects studied by Lanzelotte in [83] concerns the extensibility of the search strategy for the optimizer, demonstrated by the implementation of four different strategies: System R, Augmented Heuristic, Iterative Improvement and Simulated Annealing. Lanzelotte is especially interested in the query optimization in new systems such as oriented object and deductive DBMS, and proposes an extensible optimizer OPUS (OPtimizer for Up-to-date database Systems) [83] for these non conventional DBMS. Recently, Bizarro et al. [11] proposed "Progressive Parametric Query Optimization" which presents a novel framework to improve the performance of processing parameterized queries.

As far as parallel database systems, a synthesis dedicated to parallel relational query optimization methods and approaches [57] has been provided in section 3. In a static context [57], the most advanced works are certainly those of Garofalakis and Ioannidis [44, 45]. They extend elegantly the propositions of [23, 41, 60] where the algorithms of parallel query are based on a uni-dimensional cost model. Furthermore, [45] tackle the scheduling problem (i.e. parallelism extraction) and the resource allocation in a context, which can be multi-query by considering a multidimensional model of used resources (i.e. preemptive, and non-preemptive). The proposals of [45] seem to be the richest in terms of categories of considered resources (i.e. multi-resource allocation), exploited parallelisms, and various allocation constraints. In a dynamic context, the efforts were mainly centered on the handling of the following problems: (i) the determination and the dynamic adaptation of the intra-operation parallelism degree, (ii) the methods of resource allocation, and (iii) the dynamic query re-optimization. We identified a set of relevant parameters, mainly: search space,

strategy generation of a parallel execution plan, optimization cost for parallel execution, and cost model. These parameters allow: (i) to compare the two optimization approaches (i.e. one-phase, two-phase), and (ii) to help in the choice of an optimal exploitation of parallel optimization approaches according to the query characteristics and the shape of search space.

In a distributed database environment, static query optimization methods are focused mainly on the optimization of inter-site communication costs, by reducing the data volume transferred between sites. Dynamic query optimization methods are based on dynamic scheduling (or re-scheduling) of inter-site operators to correct the sub-optimality due to the inaccuracies of estimations and variations of available resources. The introduction of a new operator, semi-join based join [10, 25], provides certainly more flexibility to optimizers. However, it increases considerably the size of search space.

Heterogeneity and autonomy of data sources characterize data integration systems. Sources might be restricted due to the limitation of their query interfaces or certain attributes must be hidden due to privacy reasons. To handle the limited query capabilities of data sources, new mechanisms have been introduced [46, 93], such as, Dependant Join Operator which is asymmetric in nature. The asymmetry of this operator causes the search space to be restricted and raises the issue of capturing valid (feasible) execution plans [92, 93, 139].

As for the optimization methods, the community quickly noticed that the centralized optimization methods [4, 7, 14, 15, 72, 73, 74, 77 ] could not be scaled up for the reasons which are previously pointed out. So, dynamic optimization methods were decentralized by leaning, mainly, on the brokers or on the mobile agents which allow decentralizing the control and scaling up.

However, it is important to observe that the decentralized dynamic methods described in sub-section 5.3 build both two hash tables (one for each operand relation). So, they do not apply to restricted data sources. Indeed, a restricted data source returns a result, only if all attributes which are mapped to ' b ' are given.

In grid environments, which are characterized by large scale and dynamicity of nodes (system instability), distributed query optimization methods are focused on two aspects: (i) proposed execution models react to state of resources by using monitoring services [36, 49, 137] and (ii) considering different forms and types of parallelism (inter-query parallelism, intra-query parallelism).

Moreover, heterogeneity, autonomy, large scale and dynamicty of nodes raise new problems and present real challenges to design and develop acceptable cost models [1, 35, 42, 43, 99, 114, 141]. Indeed, for instance, the statistics describing the data stemming from sources and the formulae associated with the operations processed by these sources cannot be often published [35]. In a large scale environment, whatever the approach of the used cost model is (i.e. history approach [1], calibration approach [43, 141], generic approach [99]) the statistics stored in the catalog are subject to obsolescence [66], which generates large variations between parameters estimated at compile time and parameters computed at runtime. In consequence, it is not realistic to replicate a cost model on all sites. This cost model should be distributed and partially replicated [66, 58]. In an execution model based on mobile agents, a part of cost model should be embedded in mobile agents. This, ensures the autonomy of mobile joins and avoids distant interactions with the site on which was emitted the query [66].

Finally, from this state of the art, we can point out the following main characteristics of query optimization methods [98]:

- Environment: query optimization methods have designed and implemented in different environments as uni-processor, parallel, distributed, and large scale.
- Type of method: a query optimization method can be static or dynamic.
- Search Space. this space can be restricted according to the nature of the considered execution plans, the limited capabilities of data sources, and the applied search strategy.
- Nature of decision-making: can be centralized or decentralized. The decentralized dynamic optimization methods correct the sub-optimality of execution plans by decentralizing the control.
- Type of modification: can be, mainly, re-optimization or re-scheduling. When the sub-optimality of an execution plan is detected, correction could be made by re-optimization process or by a re-scheduling process. Re-optimization process: consists in producing a new execution plan for the remainder of the query [77]. The physical implementation, the scheduling and the tree structure of operators which are not yet been executed can be updated. As far as re-scheduling process, the tree structure of the remainder of the execution plan remains unchanged. But, scheduling between the operators can be modified.
- Level of modification: can occur at intra-operator level or inter-operator level. The sub-optimal execution plan can be corrected during the execution of an operator and/or at sub-query level.
- Type of event: a dynamic query optimization method can react to following events: (i) estimation errors, (ii) available memory, (iii) delays in data arrival rates, and (iv) user preferences.

These parameters allow comparing proposed optimization methods, and pointing out their advantages and weaknesses. A comparison study of dynamic optimization methods is described in detail in [98]. Furthermore, in a large scale environment, the benefits of mobile agents depending on estimation errors of temporary relation sizes, network bandwidth, and processor frequency, seem to be very promising due to their autonomy and proactive behavior.

## 8 Conclusion

Researches related to relational query optimization goes back to the 70s, and began with the publication of two papers [112, 138]. These papers and relevant applications requirements motivated a large part of the database community to focus their efforts and energies on this topic. Because of the importance and the complexity of the query optimization problem, the database community has proposed approaches, methods and techniques in different environments (uni-processor, parallel, distributed, large scale).

In this paper, we wanted to provide a survey related to evolution of query optimization methods from centralized relational database systems to data grid systems through parallel and distributed database systems and data integration (mediation)

systems. For each environment, we described some query optimization methods, and pointed out their main characteristics which allow comparing them.

## Acknowledgement

We would like to warmly thank Professor Roland Wagner for his kind invitation to write this paper.

## Permissions

57. Hameurlain, A., Morvan, F.: Parallel query optimization methods and approaches: a survey. Journal of Computers Systems Science & Engineering 19(5), 95–114 (2004)
58. Hameurlain, A., Morvan, F., El Samad, M.: Large Scale Data management in Grid Systems: a Survey. In: IEEE Intl. Conf. on Information and Communication Technologies: from Theory to Applications, pp. 1–6. IEEE CS, Los Alamitos (2008)
98. Morvan, F., Hameurlain, A.: Dynamic Query Optimization: Towards Decentralized Methods. Intl. Jour. of Intelligent Information and Database Systems (to appear, 2009)

Section 1 contains materials from [98] with kind permissions from Inderscience. Section 3 contains materials from [57] with kind permissions from CRL Publishing. Section 5 contains materials from [98] with kind permissions from Inderscience. Section 6 and 7 contain materials from [58, 98] with kind permissions from IEEE and Inderscience.

## References

1. Adali, S., Candan, K.S., Papakonstantinou, Y., Subrahmanian, V.S.: Query Caching and Optimization in Distributed Mediator Systems. In: Proc. of ACM SIGMOD Intl. Conf. on Management of Data, pp. 137–148. ACM Press, New York (1996)
2. Alpdemir, M.N., Mukherjee, A., Gounaris, A., Paton, N.W., Fernandes, A.A.A., Sakellariou, R., Watson, P., Li, P.: Using OGSA-DQP to support scientific applications for the grid. In: Herrero, P., S. Pérez, M., Robles, V. (eds.) SAG 2004. LNCS, vol. 3458, pp. 13–24. Springer, Heidelberg (2005)
3. Amsaleg, L., Franklin, M.J., Tomasic, A., Urhan, T.: Scrambling query plans to cope with unexpected delays. In: Proc. of the Fourth Intl. Conf. on Parallel and Distributed Information Systems, pp. 208–219. IEEE CS, Los Alamitos (1996)
4. Amsaleg, L., Franklin, M., Tomasic, A.: Dynamic query operator scheduling for wide-area remote access. Distributed and Parallel Databases 6(3), 217–246 (1998)
5. Antonioletti, M., et al.: The design and implementation of Grid database services in OGSA-DAI. In: Concurrency and Computation: Practice & Experience, vol. 17, pp. 357–376. Wiley InterScience, Hoboken (2005)

6. Arcangeli, J.-P., Hameurlain, A., Migeon, F., Morvan, F.: Mobile Agent Based Self-Adaptive Join for Wide-Area Distributed Query Processing. Jour. of Database Management 15(4), 25–44 (2004)
7. Avnur, R., Hellerstein, J.-M.: Eddies: Continuously Adaptive Query Processing. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, vol. 29, pp. 261–272. ACM Press, New York (2000)
8. Babu, S., Bizarro, P., De Witt, D.J.: Proactive re-optimization. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp. 107–118. ACM Press, New York (2005)
9. Bancilhon, F., Ramakrishnan, R.: An Amateur's Introduction to Recursive Query Processing Strategies. In: Proc. of the 1986 ACM SIGMOD Conf. on Management of Data, vol. 15, pp. 16–52. ACM Press, New York (1986)
10. Bernstein, P.A., Goodman, N., Wong, E., Reeve, C.L., Rothnie Jr.: Query Processing in a System for Distributed Databases (SDD-1). ACM Trans. Database Systems 6(4), 602–625 (1981)
11. Bizarro, P., Bruno, N., De Witt, D.J.: Progressive Parametric Query Optimization. IEEE Transactions on Knowledge and Data Engineering 21(4), 582–594 (2009)
12. Bonneau, S., Hameurlain, A.: Hybrid Simultaneous Scheduling and Mapping in SQL Multi-query Parallelization. In: Bench-Capon, T.J.M., Soda, G., Tjoa, A.M. (eds.) DEXA 1999. LNCS, vol. 1677, pp. 88–99. Springer, Heidelberg (1999)
13. Bose, S.K., Krishnamoorthy, S., Ranade, N.: Allocating Resources to Parallel Query Plans in Data Grids. In: Proc. of the 6th Intl. Conf. on Grid and Cooperative Computing, pp. 210–220. IEEE CS, Los Alamitos (2007)
14. Bouganim, L., Fabret, F., Mohan, C., Valduriez, P.: A dynamic query processing architecture for data integration systems. Journal of IEEE Data Engineering Bulletin 23(2), 42–48 (2000)
15. Bouganim, L., Fabret, F., Mohan, C., Valduriez, P.: Dynamic query scheduling in data integration systems. In: Proc. of the 16th Intl. Conf. on Data Engineering, pp. 425–434. IEEE CS, Los Alamitos (2000)
16. Bratbergsengen, K.: Hashing Methods and Relational Algebra Operations. In: Proc. of 10th Intl. Conf. on VLDB, pp. 323–333. Morgan Kaufmann, San Francisco (1984)
17. Brunie, L., Kosch, H.: Control Strategies for Complex Relational Query Processing in Shared Nothing Systems. SIGMOD Record 25(3), 34–39 (1996)
18. Brunie, L., Kosch, H.: Intégration d'heuristiques d'ordonnancement dans l'optimisation parallèle de requêtes relationnelles. Revue Calculateurs Parallèles, numéro spécial: Bases de données Parallèles et Distribuées 9(3), 327–346 (1997); Ed. Hermès
19. Brunie, L., Kosch, H., Wohner, W.: From the modeling of parallel relational query processing to query optimization and simulation. Parallel Processing Letters 8, 2–24 (1998)
20. Bruno, N., Chaudhuri, S.: Efficient Creation of Statistics over Query Expressions. In: Proc. of the 19th Intl. Conf. on Data Engineering, Bangalore, India, pp. 201–212. IEEE CS, Los Alamitos (2003)
21. Chaudhuri, S.: An Overview of Query Optimization in Relational Systems. In: Symposium in Principles of Database Systems PODS 1998, pp. 34–43. ACM Press, New York (1998)
22. Chawathe, S.S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J.D., Widom, J.: The TSIMMIS Project: Integration of Heterogeneous Information Sources. In: Proc. of the 10th Meeting of the Information Processing Society of Japan, pp. 7–18 (1994)

23. Chekuri, C., Hassan, W.: Scheduling Problem in Parallel Query Optimization. In: Symposium in Principles of Database Systems PODS 1995, pp. 255–265. ACM Press, New York (1995)
24. Chen, M.S., Lo, M., Yu, P.S., Young, H.S.: Using Segmented Right-Deep Trees for the Execution of Pipelined Hash Joins. In: Proc. of the 18th VLDB Conf., pp. 15–26. Morgan Kaufmann, San Francisco (1992)
25. Chiu, D.M., Ho, Y.C.: A Methodology for Interpreting Tree Queries Into Optimal Semi-Join Expressions. In: Proc. of the 1980 ACM SIGMOD, pp. 169–178. ACM Press, New York (1980)
26. Christophides, V., Cluet, S., Moerkotte, G.: Evaluating Queries with Generalized Path Expression. In: Proc. of the 1996 ACM SIGMOD, vol. 25, pp. 413–422. ACM Press, New York (1996)
27. Cole, R.L., Graefe, G.: Optimization of dynamic query evaluation plans. In: Proc. of the 1994 ACM SIGMOD, vol. 24, pp. 150–160. ACM Press, New York (1994)
28. Collet, C., Vu, T.-T.: QBF: A Query Broker Framework for Adaptable Query Evaluation. In: Christiansen, H., Hacid, M.-S., Andreasen, T., Larsen, H.L. (eds.) FQAS 2004. LNCS, vol. 3055, pp. 362–375. Springer, Heidelberg (2004)
29. Cybula, P., Kozankiewicz, H., Stencel, K., Subieta, K.: Optimization of Distributed Queries in Grid Via Caching. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 387–396. Springer, Heidelberg (2005)
30. Da Silva, V.F.V., Dutra, M.L., Porto, F., Schulze, B., Barbosa, A.C., de Oliveira, J.C.: An adaptive parallel query processing middleware for the Grid. In: Concurrence and Computation: Pratique and Experience, vol. 18, pp. 621–634. Wiley InterScience, Hoboken (2006)
31. Date, C.J.: An Introduction to Database Systems, 6th edn. Addison-Wesley, Reading (1995)
32. Deshpande, A., Hellerstein, J.-M.: Lifting the Burden of History from Adaptive Query Processing. In: Proc. of the 13th Intl. Conf. on VLDB, pp. 948–959. Morgan Kaufmann, San Francisco (2004)
33. De Witt, D.J., Kabra, N., Luo, J., Patel, J.M., Yu, J.B.: Client-Server Paradise. In: Proc. of the 20th VLDB Conf., pp. 558–569. Morgan Kaufmann, San Francisco (1994)
34. Dinquel, J.: Network Architectures for Cluster Computing. Technical Report 572, CECS, California State University (2000)
35. Du, W., Krishnamurthy, R., Shan, M.-C.: Query Optimization in a Heterogeneous DBMS. In: Proc. of the 18th Intl. Conf. on VLDB, pp. 277–291. Morgan Kaufmann, San Francisco (1992)
36. El Samad, M., Gossa, J., Morvan, F., Hameurlain, A., Pierson, J.-M., Brunie, L.: A monitoring service for large-scale dynamic query optimisation in a grid environment. Intl. Jour. of Web and Grid Services 4(2), 222–246 (2008)
37. Evrendilek, C., Dogac, A., Nural, S., Ozcan, F.: Multidatabase Query Optimization. Journal of Distributed and Parallel Databases 5(1), 77–113 (1997)
38. Foster, I.: The Grid: A New Infrastructure for 21st Century Science. Physics Today 55(2), 42–56 (2002)
39. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (2004)
40. Fuggetta, A., Picco, G.-P., Vigna, G.: Understanding Code Mobility. IEEE Transactions on Software Engineering 24(5), 342–361 (1998)

41. Ganguly, S., Hasan, W., Krishnamurthy, R.: Query Optimization for Parallel Execution. In: Proc. of the 1992 ACM SIGMOD int'l. Conf. on Management of Data, vol. 21, pp. 9–18. ACM Press, San Diego (1992)
42. Ganguly, S., Goel, A., Silberschatz, A.: Efficient and Accurate Cost Models for Parallel Query Optimization. In: Symposium in Principles of Database Systems PODS 1996, pp. 172–182. ACM Press, New York (1996)
43. Gardarin, G., Sha, F., Tang, Z.-H.: Calibrating the Query Optimizer Cost Model of IRO-DB, an Object-Oriented Federated Database System. In: Proc. of 22nd Intl. Conf. on VLDB, pp. 378–389. Morgan Kaufmann, San Francisco (1996)
44. Garofalakis, M.N., Ioannidis, Y.E.: Multi-dimensional Resource Scheduling for Parallel Queries. In: Proc. of the 1996 ACM SIGMOD intl. Conf. on Management of Data, vol. 25, pp. 365–376. ACM Press, New York (1996)
45. Garofalakis, M.N., Ioannidis, Y.E.: Parallel Query Scheduling and Optimization with Time- and Space - Shared Resources. In: Proc. of the 23rd VLDB Conf., pp. 296–305. Morgan Kaufmann, San Francisco (1997)
46. Goldman, R., Widom, J.: WSQ/DSQ: A practical approach for combined querying of databases and the web. In: Proc. of ACM SIGMOD Conf., pp. 285–296. ACM Press, New York (2000)
47. Gounaris, A., Paton, N.W., Fernandes, A.A.A., Sakellariou, R.: Adaptive Query Processing: A Survey. In: Eaglestone, B., North, S.C., Poulovassilis, A. (eds.) BNCOD 2002. LNCS, vol. 2405, pp. 11–25. Springer, Heidelberg (2002)
48. Gounaris, A., Paton, N.W., Sakellariou, R., Fernandes, A.A.A.: Adaptive Query Processing and the Grid: Opportunities and Challenges. In: Proc. of the 15th Intl. Dexa Workhop, pp. 506–510. IEEE CS, Los Alamitos (2004)
49. Gounaris, A., Sakellariou, R., Paton, N.W., Fernandes, A.A.A.: Resource Scheduling for Parallel Query Processing on Computational Grids. In: Proc. of the 5th IEEE/ACM Intl. Workshop on Grid Computing, pp. 396–401 (2004)
50. Gounaris, A., Smith, J., Paton, N.W., Sakellariou, R., Fernandes, A.A.A., Watson, P.: Adapting to Changing Resource Performance in Grid Query. In: Pierson, J.-M. (ed.) VLDB DMG 2005. LNCS, vol. 3836, pp. 30–44. Springer, Heidelberg (2006)
51. Graefe, G.: Query Evaluation Techniques for Large Databases. ACM Computing Survey 25(2), 73–170 (1993)
52. Graefe, G.: Volcano - An Extensible and Parallel Query Evaluation System. IEEE Trans. Knowl. Data Eng. 6(1), 120–135 (1994)
53. Haas, L.M., Kossmann, D., Wimmers, E.L., Yang, J.: Optimizing Queries Across Diverse Data Sources. In: Proc. of 23rd Intl. Conf. on VLDB, pp. 276–285. Morgan Kaufmann, San Francisco (1997)
54. Hameurlain, A., Bazex, P., Morvan, F.: Traitement parallèle dans les bases de données relationnelles: concepts, méthodes et applications. Cépaduès Editions (1996)
55. Hameurlain, A., Morvan, F.: An Overview of Parallel Query Optimization in Relational Systems. In: 11th Intl Worshop on Database and Expert Systems Applications, pp. 629–634. IEEE CS, Los Alamitos (2000)
56. Hameurlain, A., Morvan, F.: CPU and incremental memory allocation in dynamic parallelization of SQL queries. Journal of Parallel Computing 28(4), 525–556 (2002)
57. Hameurlain, A., Morvan, F.: Parallel query optimization methods and approaches: a survey. Journal of Computers Systems Science & Engineering 19(5), 95–114 (2004)
58. Hameurlain, A., Morvan, F., El Samad, M.: Large Scale Data management in Grid Systems: a Survey. In: IEEE Intl. Conf. on Information and Communication Technologies: from Theory to Applications, pp. 1–6. IEEE CS, Los Alamitos (2008)

59. Han, W.-S., Ng, J., Markl, V., Kache, H., Kandil, M.: Progressive optimization in a shared-nothing parallel database. In: Proc.of the ACM SIGMOD Intl. Conf. on Management of Data, pp. 809–820 (2007)
60. Hasan, W., Motwani, R.: Optimization Algorithms for Exploiting the Parallelism - Communication Tradeoff in Pipelined Parallelism. In: Proc. of the 20th int'l. Conf. on VLDB, pp. 36–47. Morgan Kaufmann, San Francisco (1994)
61. Hasan, W., Florescu, D., Valduriez, P.: Open Issues in Parallel Query Optimization. SIGMOD Record 25(3), 28–33 (1996)
62. Hellerstein, J.M., Franklin, M.J.: Adaptive Query Processing: Technology in Evolution. Bulletin of Technical Committee on Data Engineering 23(2), 7–18 (2000)
63. Hong, W.: Exploiting Inter-Operation Parallelism in XPRS. In: Proc. ACM SIGMOD Conf. on Management of Data, pp. 19–28. ACM Press, New York (1992)
64. Howes, T., Smith, M.C., Good, G.S., Howes, T.A., Smith, M.: Understanding and Deploying LDAP Directory Services. MacMillan, Basingstoke (1999)
65. Hu, N., Wang, Y., Zhao, L.: Dynamic Optimization of Sub query Processing in Grid Database, Natural Computation. In: Proc of the 3rd Intl Conf. on Natural Computation, vol. 5, pp. 8–13. IEEE CS, Los Alamitos (2007)
66. Hussein, M., Morvan, F., Hameurlain, A.: Embedded Cost Model in Mobile Agents for Large Scale Query Optimization. In: Proc. of the 4th Intl. Symposium on Parallel and Distributed Computing, pp. 199–206. IEEE CS, Los Alamitos (2005)
67. Hussein, M., Morvan, F., Hameurlain, A.: Dynamic Query Optimization: from Centralized to Decentralized. In: 19th Intl. Conf. on Parallel and Distributed Computing Systems, ISCA, pp. 273–279 (2006)
68. Ioannidis, Y.E., Wong, E.: Query Optimization by Simulated Annealing. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp. 9–22. ACM Press, New York (1987)
69. Ioannidis, Y.E., Kang, Y.C.: Randomized Algorithms for Optimizing Large Join Queries. In: Proc of the 1990 ACM SIGMOD Conf. on the Manag. of Data, vol. 19, pp. 312–321 (1990)
70. Ioannidis, Y.E., Christodoulakis, S.: On the Propagation of Errors in the Size of Join Results. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp. 268–277. ACM Press, New York (1991)
71. Ioannidis, Y.E., Ng, R.T., Shim, K., Sellis, T.K.: Parametric Query Optimization. In: 18th Intl. Conf. on VLDB, pp. 103–114. Morgan Kaufmann, San Francisco (1992)
72. Ives, Z.-G., Florescu, D., Friedman, M., Levy, A.Y., Weld, D.S.: An adaptive query execution system for data integration. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp. 299–310. ACM Press, New York (1999)
73. Ives, Z.-G., Levy, A.Y., Weld, D.S., Florescu, D., Friedman, M.: Adaptive query processing for internet applications. Journal of IEEE Data Engineering Bulletin 23(2), 19–26 (2000)
74. Ives, Z.-G., Halevy, A.-Y., Weld, D.-S.: Adapting to Source Properties in Processing Data Integration Queries. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp. 395–406. ACM Press, New York (2004)
75. Jarke, M., Koch, J.: Query Optimization in Database Systems. ACM Comput. Surv. 16(2), 111–152 (1984)
76. Jones, R., Brown, J.: Distributed query processing via mobile agents (1997), `http://www.cs.umd.edu/~rjones/paper.html`

77. Kabra, N., Dewitt, D.J.: Efficient Mid - Query Re-Optimization of Sub-Optimal Query Execution Plans. In: Proc. of the ACM SIGMOD intl. Conf. on Management of Data, vol. 27, pp. 106–117. ACM Press, New York (1998)
78. Kabra, N., De Witt, D.J.: OPT++: An Object-Oriented Implementation for Extensible Database Query Optimization. VLDB Journal 8, 55–78 (1999)
79. Khan, M.F., Paul, R., Ahmed, I., Ghafoor, A.: Intensive Data Management in Parallel Systems: A Survey. Distributed and Parallel Databases 7, 383–414 (1999)
80. Khan, L., Mcleod, D., Shahabi, C.: An Adaptive Probe-Based Technique to Optimize Join Queries in Distributed Internet Databases. Journal of Database Management 12(4), 3–14 (2001)
81. Kosch, H.: Managing the operator ordering problem in parallel databases. Future Generation Computer Systems 16(6), 665–676 (2000)
82. Kossmann, D.: The State of the Art in Distributed Query Processing. ACM Computing Surveys 32(4), 422–469 (2000)
83. Lanzelotte, R.S.G.: OPUS: an extensible Optimizer for Up-to-date database Systems. Ph-D Thesis, Computer Science, PUC-RIO, available at INRIA, Rocquencourt, n° TU-127 (1990)
84. Lanzelotte, R.S.G., Valduriez, P.: Extending the Search Strategy in a Query Optimizer. In: Proc. of the Int'l Conf. on VLDB, pp. 363–373. Morgan Kaufmann, San Francisco (1991)
85. Lanzelotte, R.S.G., Zaït, M., Gelder, A.V.: Measuring the effectiveness of optimization. Search Strategies. In: BDA 1992, Trégastel, pp. 162–181 (1992)
86. Lanzelotte, R.S.G., Valduriez, P., Zaït, M.: On the Effectiveness of Optimization Search Strategies for Parallel Execution Spaces. In: Proc. of the Intl Conf. on VLDB, pp. 493–504. Morgan Kaufmann, San Francisco (1993)
87. Lazaridis, I., Mehrotra, S.: Optimization of multi-version expensive predicates. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp. 797–808. ACM Press, New York (2007)
88. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying Heterogeneous Information Sources Using Source Descriptions. In: Proc. of the Intl. Conf. on VLDB, pp. 251–262. Morgan Kaufmann, San Francisco (1996)
89. Liu, S., Karimi, H.A.: Grid query optimizer to improve query processing in grids. Future Generation Computer Systems 24(5), 342–353 (2008)
90. Lu, H., Ooi, B.C., Tan, K.-L.: Query Processing in Parallel Relational Database Systems. IEEE CS Press, Los Alamitos (1994)
91. Mackert, L.F., Lohman, G.M.: R* Optimizer Validation and Performance Evaluation for Distributed Queries. In: Proc. of the 12th Intl. Conf. on VLDB, pp. 149–159 (1986)
92. Manolescu, I.: Techniques d'optimisation pour l'interrogation des sources de données hétérogènes et distribuées, Ph-D Thesis, Université de Versailles Saint-Quentin-en-Yvlenies, France (2001)
93. Manolescu, I., Bouganim, L., Fabret, F., Simon, E.: Efficient querying of distributed resources in mediator systems. In: Meersman, R., Tari, Z., et al. (eds.) CoopIS 2002, DOA 2002, and ODBASE 2002. LNCS, vol. 2519, pp. 468–485. Springer, Heidelberg (2002)
94. Marzolla, M., Mordacchini, M., Orlando, S.: Peer-to-Peer for Discovering resources in a Dynamic Grid. Jour. of Parallel Computing 33(4-5), 339–358 (2007)
95. Mehta, M., Dewitt, D.J.: Managing Intra-Operator Parallelism in Parallel Database Systems. In: Proc. of the 21th Intl. Conf. on VLDB, pp. 382–394 (1995)
96. Mehta, M., Dewitt, D.J.: Data Placement in Shared-Nothing Parallel Database Systems. The VLDB Journal 6, 53–72 (1997)

97. Morvan, F., Hussein, M., Hameurlain, A.: Mobile Agent Cooperation Methods for Large Scale Distributed Dynamic Query Optimization. In: Proc. of the 14th Intl. Workshop on Database and Expert Systems Applications, pp. 542–547. IEEE CS, Los Alamitos (2003)
98. Morvan, F., Hameurlain, A.: Dynamic Query Optimization: Towards Decentralized Methods. Intl. Jour. of Intelligent Information and Database Systems (to appear, 2009)
99. Naacke, H., Gardarin, G., Tomasic, A.: Leveraging Mediator Cost Models with Heterogeneous Data Sources. In: Proc. of the 14th Intl. Conf. on Data Engineering, pp. 351–360. IEEE CS, Los Alamitos (1998)
100. Ono, K., Lohman, G.M.: Measuring the Complexity of Join Enumeration in Query Optimization. In: Proc. of the Int'l Conf. on VLDB, pp. 314–325. Morgan Kaufmann, San Francisco (1990)
101. Ozakar, B., Morvan, F., Hameurlain, A.: Mobile Join Operators for Restricted Sources. Mobile Information Systems: An International Journal 1(3), 167–184 (2005)
102. Ozcan, F., Nural, S., Koksal, P., Evrendilek, C., Dogac, A.: Dynamic query optimization in multidatabases. Data Engineering Bulletin CS 20(3), 38–45 (1997)
103. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 2nd edn. Prentice-Hall, Englewood Cliffs (1999)
104. Pacitti, E., Valduriez, P., Mattoso, M.: Grid Data Management: Open Problems and News Issues. Intl. Journal of Grid Computing 5(3), 273–281 (2007)
105. Paton, N.W., Chávez, J.B., Chen, M., Raman, V., Swart, G., Narang, I., Yellin, D.M., Fernandes, A.A.A.: Autonomic query parallelization using non-dedicated computers: an evaluation of adaptivity options. VLDB Journal 18(1), 119–140 (2009)
106. Porto, F., da Silva, V.F.V., Dutra, M.L., Schulze, B.: An Adaptive Distributed Query Processing Grid Service. In: Pierson, J.-M. (ed.) VLDB DMG 2005. LNCS, vol. 3836, pp. 45–57. Springer, Heidelberg (2006)
107. Rahm, E., Marek, R.: Dynamic Multi-Resource Load Balancing in Parallel Database Systems. In: Proc. of the 21st VLDB Conf., pp. 395–406 (1995)
108. Rajaraman, A., Sagiv, Y., Ullman, J.D.: Answering queries using templates with binding patterns. In: The Proc. of ACM PODS, pp. 105–112. ACM Press, New York (1995)
109. Raman, V., Deshpande, A., Hellerstein, J.-M.: Using State Modules for Adaptive Query Processing. In: Proc. of the 19th Intl. Conf. on Data Engineering, pp. 353–362. IEEE CS, Los Alamitos (2003)
110. Sahuguet, A., Pierce, B., Tannen, V.: Distributed Query Optimization: Can Mobile Agents Help? (2000), http://www.seas.upenn.edu/~gkarvoun/dragon/publications/sahuguet/
111. Schneider, D., Dewitt, D.J.: Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines. In: Proc. of the 16th VLDB Conf., pp. 469–480. Morgan Kaufmann, San Francisco (1990)
112. Selinger, P.G., Astrashan, M., Chamberlin, D., Lorie, R., Price, T.: Access Path Selection in a Relational Database Management System. In: Proc. of the 1979 ACM SIGMOD Conf. on Management of Data, pp. 23–34. ACM Press, New York (1979)
113. Selinger, P.G., Adiba, M.E.: Access Path Selection in Distributed Database Management Systems. In: Proc. Intl. Conf. on Data Bases, pp. 204–215 (1980)
114. Slimani, Y., Najjar, F., Mami, N.: An Adaptable Cost Model for Distributed Query Optimization on the Grid. In: Meersman, R., Tari, Z., Corsaro, A. (eds.) OTM-WS 2004. LNCS, vol. 3292, pp. 79–87. Springer, Heidelberg (2004)

115. Smith, J., Gounaris, A., Watson, P., Paton, N.W., Fernandes, A.A.A., Sakellariou, R.: Distributed Query Processing on the Grid. In: Parashar, M. (ed.) GRID 2002. LNCS, vol. 2536, pp. 279–290. Springer, Heidelberg (2002)
116. Soe, K.M., New, A.A., Aung, T.N., Naing, T.T., Thein, N.L.: Efficient Scheduling of Resources for Parallel Query Processing on Grid-based Architecture. In: Proc. of the 6th Asia-Pacific Symposium, pp. 276–281. IEEE CS, Los Alamitos (2005)
117. Stillger, M., Lohman, G.M., Markl, V., Kandil, M.: LEO - DB2's LEarning Optimizer. In: Proc.of 27th Intl. Conf. on Very Large Data Bases, pp. 19–28. Morgan Kaufmann, San Francisco (2001)
118. Stonebraker, M., Katz, R.H., Paterson, D.A., Ousterhout, J.K.: The Design of XPRS. In: Proc. of the 4th VLDB Conf., pp. 318–330. Morgan Kaufmann, San Francisco (1988)
119. Stonebraker, M., Aoki, P.M., Litwin, W., Pfeffer, A., Sah, A., Sidell, J., Staelin, C., Yu, A.: Mariposa: A Wide-Area Distributed Database System. VLDB Jour. 5(1), 48–63 (1996)
120. Stonebraker, M., Hellerstein, J.M.: Readings in Database Systems, 3rd edn. Morgan Kaufmann, San Francisco (1998)
121. Swami, A.: Optimization of large join queries. Technical report, Software Techonology Laboratory, H-P Laboratories, Report STL-87-15 (1987)
122. Swami, A.N., Gupta, A.: Optimization of Large Join Queries. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp. 8–17. ACM Press, New York (1988)
123. Swami, A.N.: Optimization of Large Join Queries: Combining Heuristic and Combinatorial Techniques. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp. 367–376 (1989)
124. Tan, K.L., Lu, H.: A Note on the Strategy Space of Multiway Join Query Optimization Problem in Parallel Systems. SIGMOD Record 20(4), 81–82 (1991)
125. Taniar, D., Leung, C.H.C.: Query execution scheduling in parallel object-oriented databases. Information & Software Technology 41(3), 163–178 (1999)
126. Taniar, D., Leung, C.H.C., Rahayu, J.W., Goel, S.: High Performance Parallel Database Processing and Grid Databases. John Wiley & Sons, Chichester (2008)
127. Tomasic, A., Raschid, L., Valduriez, P.: Scaling Heterogeneous Databases and the Design of Disco. In: Proc. of the 16th Intl. Conf. on Distributed Computing Systems, pp. 449–457. IEEE CS, Los Alamitos (1996)
128. Tomasic, A., Raschid, L., Valduriez, P.: Scaling Access to Heterogeneous Data Sources with DISCO. IEEE Trans. Knowl. Data Eng. 10(5), 808–823 (1998)
129. Trunfio, P., et al.: Peer-to-Peer resource discovery in Grids: Models and systems. Future Generation Computer Systems 23(7), 864–878 (2007)
130. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, vol. I. Computer Science Press (1988)
131. Urhan, T., Franklin, M.: XJoin: A reactively-scheduled pipelined join operator. IEEE Data Engineering Bulletin 23(2), 27–33 (2000)
132. Urhan, T., Franklin, M.: Dynamic pipeline scheduling for improving interactive query performance. In: Proc.of 27th Intl. Conf. on VLDB, pp. 501–510. Morgan Kaufmann, San Francisco (2001)
133. Valduriez, P.: Semi-Join Algorithms for Distributed Database Machines. In: Proc. of the 2nd Intl. Symposium on Distributed Data Bases, pp. 22–37. North-Holland Publishing Company, Amsterdam (1982)
134. Valduriez, P., Gardarin, G.: Join and Semijoin Algorithms for a Multiprocessor Database Machine. ACM Trans. Database Syst. 9(1), 133–216 (1984)

135. Wohrer, A., Brezany, P., Tjoa, A.M.: Novel mediator architectures for Grid information systems. Future Generation Computer Systems, 107–114 (2005)
136. Wiederhold, G.: Mediators in the Architecture of Future Information Systems. Journal of IEEE Computer 25(3), 38–49 (1992)
137. Wolski, R., Spring, N.T., Hayes, J.: The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Journal of Future Generation Computing Systems 15(5-6), 757–768 (1999)
138. Wong, E., Youssefi, K.: Decomposition: A Strategy for Query Processing. ACM Transactions on Database Systems 1, 223–241 (1976)
139. Yerneni, R., Li, C., Ullman, J.D., Garcia-Molina, H.: Optimizing Large Join Queries in Mediation Systems. In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 348–364. Springer, Heidelberg (1998)
140. Zhou, Y., Ooi, B.C., Tan, K.-L., Tok, W.H.: An adaptable distributed query processing architecture. Data & Knowledge Engineering 53(3), 283–309 (2005)
141. Zhu, Q., Motheramgari, S., Sun, Y.: Cost Estimation for Queries Experiencing Multiple Contention States in Dynamic Multidatabase Environments. Journal of Knowledge and Information Systems Publishers 5(1), 26–49 (2003)
142. Ziane, M., Zait, M., Borlat-Salamet, P.: Parallel Query Processing in DBS3. In: Proc of the 2nd Intl. Conf. on Parallel and Distributed Information Systems, pp. 93–102. IEEE CS, Los Alamitos (1993)