EFFECTIVE IMPLEMENTATION OF QUERY OPTIMIZATION THROUGH PERFORMANCE TUNING TECHNIQUES ON WEB

P.JYOTSNA¹, P.SUNIL KUMAR REDDY¹, P.GOVINDARAJULU²

¹² Dept of Computer science, S.V.University, Tirupati
¹ pg.sunilkumar@gmail.com , ¹¹ pl_jyotsna@yahoo.co.in , ² pgovindarajulu@yahoo.com

ABSTRACT
This study focuses on the advent of the Internet and the Web and to connect information sources across all types of boundaries where information sources include databases, XML documents, and other unstructured sources. This study highlights uniformly querying those information sources and implementing Query Optimization techniques. Since the information is scattered in the Web, querying the sources forms the basis to improve efficiency to retrieve information which can be done through Query Optimization. In this paper, we survey the current research on fundamental problems to efficiently process queries over Web data integration systems. Optimization techniques handle different dimensions of information like volatility, heterogeneity and autonomy which form the case for information retrieval. This study outlines framework for evaluating such information through Query optimization techniques.

Keywords: QueryOptimization, DataIntegration, OODBMS, Informationretrieval, Database Management Systems, Performance tuning, Object Query Language, Stack based approach, Relational databases.

1. INTRODUCTION
Wide spread deployment and use of web plays a key role behind the growth and expansion of Information in the Internet which comprises high speed networking and communication middleware. Globalization has spurred the development of tools and aids to navigate and share information in corporate intranets that were previously accessible online only in a restrictive way and at prohibitive costs. This highlighted the role of the relational database management system (RDBMS) as a key enabling technology. DBMSs are currently the technology of choice for modeling, storing, managing, and efficiently querying large amounts of information[24]. They also provide functions for protecting data integrity, facilitating reliable and flexible data access and manipulation, synchronizing concurrent accesses from applications, and securing data. Web provides access to a variety of data including multimedia. retrieval techniques like inverted indices, performance tuning allow efficient keyword-based access to text, largely enabled access to the exponentially growing Web. This extends beyond keyword-based search so as to provide access to volatile information which is achieved by using databases to model Web pages, information could be extracted to dynamically build a schema against which users could submit SQL-like queries. In addition to this, XML can be used adapted for data representation, which adds database constructs to HTML to provide richer, queriable data types. Databases require better models and tools for describing data semantics and specifying Metadata due to the large number of cooperating databases complicates autonomy and heterogeneity issues. Techniques for automatic data and metadata extraction and classification like ontologies play crucial role in representing semantic web related information.

Query languages and query processing and optimization techniques need to be extended to exploit semantic information. Users also need adaptive systems to help them explore the Web and discover interfaces that support different query and search paradigms. Data dissemination techniques and notification services must be developed to enable effective data delivery services. In spite of providing high performance based secured metrics in applications like Ecommerce, there is a need to provide support to access and deploy information on web uniformly and efficiently. Issues related to building data integration infrastructures include research in distributed database systems and multidatabase systems[3]. Databases and applications evolve in an environment characterized by a larger number of resources, high dynamics, easier interconnection, higher distribution, (quasi) lack of organizational and control structures, larger number of clients, etc. In addition to these characteristics, goals and means to achieve coordinated access to databases on the Web need to be reevaluated. Query optimization takes a central stage in data integration systems on the Web. Web’s characteristics make the task of enabling efficient querying even more difficult. A major difficulty in optimizing queries on the Web is that once a query is submitted to a specific information source, control over its execution is no longer possible. Further, that information source may exhibit a different behavior from what has been initially assumed, thus impairing predictions. As a result, traditional optimization techniques that rely heavily on statistical information may be hardly applicable. Query optimization on the Web may also span a larger spectrum of criteria than those in the classical cost model. Web’s volatility and highly dynamic nature are a challenge when the expectation is that queries always return results. All information sources do not provide the
same query capabilities. The query processor needs to make sure that the generated query execution plan is feasible with respect to these limitations. In this study, we survey queries processing and optimization in Web data integration systems. We also present a classification of the different presented techniques and a comprehensive framework to evaluate them. Most of the described systems adopt the mediator approach [49] for data integration. Those are systems that match information requests from consumers, individual users or applications, to information providers[1]. This survey classifies the different systems according to the focus of their query optimization approaches. We overview cost-based optimization techniques, adaptive query optimization, quality-based optimization, and query optimization in presence of sources with limited capabilities. In general, cost-based optimization techniques extend the classical cost model through various means such as estimation, caching, etc. Adaptive schemes addresses efficiency in presence of unpredictable events. These techniques focuses on optimizing queries while making sure that their execution is possible.

2. PRE-WEB DATA INTEGRATION

We outline, major work involved in database integration and query optimization in the Pre-Web and how to address Query optimization. Given a declarative query, e.g., in SQL, there are several execution plans that can be used to produce its results. These different plans answer the same query and are equivalent in terms of their final output but may differ according to different performance parameters like response time and resource use. So Query optimization involves selecting the best query execution plan while designing a module. The role of the optimizer is to determine a query execution plan that minimizes an objective cost function. The optimization problem can be described abstractly as follows[19]: Given a query Q, an execution space E that computes Q, and a cost function c defined over E, find an execution e in E (the subset of E that computes Q) of minimum cost: \[ \text{min}_{e \in E(Q)} c(e) \]. An optimizer can be characterized by three dimensions: (i) Execution space that captures the underlying execution model and defines the alternative executions. (ii) Cost model to predict the cost of an execution, and (iii) Search strategy that enumerates the execution plans and select the best one.

Query optimization strategies are classified in three main categories[36]:
• Heuristic-based. Heuristic rules are used to re-arrange the different operations in a query execution plan. For example, to minimize the size of intermediate results.
• Cost-based. The costs of different strategies are estimated and the best one is selected in order to minimize the objective cost function. For example, the number of I/Os.
• Hybrid. Heuristic rules and cost estimates are combined together.

In most cases, the focus is on the join operation which is the most costly operation.

2.1. Data integration approaches

Data integration generally implies uniform and transparent access to data managed by multiple databases. A mechanism to achieve that goal is through an integrated schema that involves all or parts of the component schemas. This is classified into three categories: global schema integration, federated databases[22], and multidatabase language approach[24]. Global schema integration was one of the first attempts at data sharing and it is based on the complete integration of multiple databases to provide a single view (global schema).

Information sharing occurs through import and export schemas. The multidatabase language approach is intended for users who do not use a predefined global or partial schema. Preexisting heterogeneous local databases are usually integrated without modifications. Information stored in different databases may be redundant, heterogeneous, and inconsistent. The aim of a multidatabase language is to provide constructs that perform queries involving several databases at the same time but problem this approach is the lack of distribution and location transparency, as users have to a-priori find the right information in a potentially large network of databases. Users are responsible for understanding schemas, and detecting and resolving semantic conflicts. In this approach, users are faced with the issues that consist of finding the relevant information in multiple databases, understanding each individual database schema, detecting and resolving semantic conflicts, and performing view integration.

2.2. Query Optimization

Query optimization plays vital role in heterogeneous distributed databases systems. Lack of statistical information from participating databases prevented a direct application of techniques developed for homogeneous systems. Different techniques have been proposed to overcome the lack of statistical information. A number of heuristics are used to optimize queries. In CORDS [53], sampling and probing are applied to local databases to estimate costs by gathering statistical information. Some special queries are executed on top of the participating databases and their behavior and results are recorded. This process is repeated as frequently as needed. Query processing at the global level is then conducted using the gathered information along with a classical cost-based model. A statistical scheme is used at runtime to determine ad select the less costly and more selective inter-site operation between currently available partial results. Each inter-site operation is
assigned with a weight that includes its cost, selectivity, and transfer cost. Operations are scheduled if their weights do not exceed a threshold value computed each time a partial result is available.

### 2.2.1 Query Optimization in Object Oriented Databases:

In order to Optimize Queries in Object Oriented Databases, Object Query Language (OQL) is used which allows to express queries explicitly in the code. Object Oriented Data Base (OODB) has all the functional features, functionality of a relational database system and is abused an Object Oriented Programming language interface. This uses stack based approach responsible naming-scoping-binding principle[45] and is based on Query rewriting. These are Collectively known as Independent subqueries. In order to Recover limitations of RDBMS, Object Oriented DBMS came into existence. An OODBMS is the result of combining object oriented programming principles with database management principles and supports ACID properties (Atomicity, Consistency, Isolation and Durability). OODB [12] is a system which supports functionality of RDBMS with object oriented features like creation of user defined datatypes, object identifiers and has ability to manage objects persistently. Following figure illustrates Query optimization in OORDBMS. As shown in the below figure, first step is Rewrite step consists in a syntactic and semantic rewrite of the query in the goal to determine simpler equivalent queries. The result of this step is the generation of a query graph. Ordering operations step is takes place in two phases: generation and assessment of the plans which determined in the first phase. Execution step permits to choose the optimal execution plan and to execute it.

![Fig: Query Optimization Steps in OODBMS](image)

**Stack based Approach (SBA) in OODBMS:**

OODBMS uses Stack based approach (SBA) for Query Optimization. SBA allows [45] to precisely determine the semantics of query languages in relation with object oriented concepts, with imperative programming constructs and with programming abstraction, including procedures, functional procedures views, modules etc. Following are the features of SBA:

(i) Naming-scoping-binding principle is followed, which means that each name occurring in a query is bound to the appropriate run-time entity (an object, attribute, method, parameter, etc.) depending upon the scope for the name.

(ii) One of its basic mechanisms is an environmental stack (ENVS). The stack is responsible for scope control, for binding names, parameter passing and procedure calls. ENVS is also responsible for processing non-algebraic query operators.

(iii) The object relativity principle is assumed, i.e. object on any hierarchy level have same formal properties and are treated uniformly. This principle simplifies semantic considerations in developing query optimization methods.

(iv) Each run-time entity that can be separately bound, inserted, updated, deleted etc. must possess a unique internal identifier.

**SBA Architecture**

SBA consists of sections that are sets of binders. Binder is a concept that allows us to explain and describe various naming issues that occur in object models and programming languages. At the beginning, the ENVS consist of single section containing binders to all root database objects. During query evaluation the stack is growing and shrinking according to query nesting. Here we assume a small database schema shown in below figure. It defines five classes as Person, Professor, Student, Lecture and Faculty. Names of classes are followed by cardinality.

Number sand Queries are combined by operators. Binary operator is subdivided into algebraic and non-algebraic. An operator is algebraic if it does not modify the state of ENVS. The algebraic operators include numerical and string operators and comparisons, Boolean and, or, not, aggregate function, and sequence operators and comparisons, structure constructor, etc. The operator group as names the entire query result, while as names each element in a sequence or bag returned by the query. If q1 Δ q2 be a query consisting of two subqueries connected by a binary algebraic operator Δ. The procedure takes q1 and pushes its result onto top of QERS then does the same with q2, performs Δ with two top Query values and finally removes top of Query value twice and pushes the final result onto top.

**Merits of SBA**
SBA simplifies Optimization process since the subqueries are independent of each other and this is based on concept that, if none of the name in subquery is bound in the ENVS section opened by the non-algebraic operator currently being evaluated, then this subquery is independent of this operator. Subqueries are called independent if they can be evaluated outside loops implied by the non-algebraic query operators. Such subqueries are worth analyzing because they usually imply optimization possibilities and this uses a special technique called the method of independent subqueries to optimize queries. Technically, it consists in analyzing in which sections particular names occurring in a query are bound. It turns out that if none of the names in given subquery is bound in the scope opened by the non-algebraic operator currently being evaluated, then that subquery can be evaluated earlier than it results from its textual place in the query it is a part of. The method modifies the textual form of a query so that all its subqueries will be evaluated as soon as possible.

Consider the above example to describe the merits of SBA. 

Consider a simple query “Get lectures whose credits are greater than the credits of physics” Lecture where credits > ((Lecture where subject = “physics”).credits) (1) Here the subquery returning the credits of physics: (Lecture where subject = “physics”).credits (2) Is evaluated for each Lecture object existing in the database, while it is enough to calculate it just once, because its evaluation gives the same result every time. We can say that this subquery is independent of its direct non-algebraic operator. Let us see how the number are assigned to query, Lecture where credits > (1, 1) 2 (2,2) ((Lecture where subject = “physics”).credits) (2,1) 3 (3,3) 3 (3,3) As shown above none of the names in subquery (2) is bound in section 2 opened by external where, therefore subquery (2) is independent of that operator and can be calculated before it opens its section. To express it in the textual form of the query, the independent subquery is factored out this is done in following ways, A new unique auxiliary name is chosen. It will be used as the name of the result of the independent subquery (2).

Then the subquery (2) is named by the as operator, put before the entire subquery (1) of the non-algebraic operator it is independent of, and connected to the rest of the query (i.e. 3) by a dot operator.

Finally, the auxiliary name is put in the previous place of (2) this sub query. After factoring sub query (2) out, query (1) will be rewrite as ((Lecture where subject = “physics”).credits) as c). (1,1) 2 (2,2) 2 (2,2) 2 Lecture where credits > c (3) (2,1) 3 (3,3) 3 (3,2). Where c is the auxiliary name. Now sub query (2) is evaluated before its result is used and the auxiliary name c naming its result makes it possible to read this result. The method of independent sub query is quite sophisticated , it recursively traversed a query abstract syntax tree(AST) to find the largest sub query which is independent of the currently evaluated non-algebraic operator.

2.2.2 Query Optimization in Relational Databases:

The query optimizer is responsible for generating the input for the execution engine. It takes a parsed representation of a SQL query [47] as input and is responsible for generating an efficient execution plan for the given SQL query from the space of possible execution plans. The task of an optimizer involves large number of possible operator trees:

- The algebraic representation of the given query can be transformed into many other logically equivalent algebraic representations: e.g., Join (A,B,C) = Join (B,C,A) where A, B and C represent table names.
- For a given algebraic representation, there may be many operator trees that implement the algebraic expression, e.g., typically there are several join algorithms supported in a database system. Query Optimization in Relational databases involves following steps:
  1. Creation of Query space plans.
  2. A cost estimation technique so that a cost may be assigned to each plan in the search space. Intuitively, this is an estimation of the resources needed for the execution of the plan.
  3. An enumeration algorithm that can search through the execution space. Optimizer is one where the search space includes plans that have low cost and the costing technique is accurate and the enumeration algorithm is efficient.

SYSTEM-R OPTIMIZER illustrates Query optimization in relational databases.
The System-R project is used for query optimization of relational systems. The ideas in [44] have been incorporated in many commercial optimizers continue to be remarkably relevant. I will present a subset of those important ideas here in the context of Select-Project-Join (SPJ) queries. The class of SPJ queries is closely related to and encapsulates conjunctive queries, which are used in Database Theory. The search space for the System-R optimizer in SPJ query consists of operator trees correspond to linear sequence of Join operations and These sequences are logically equivalent because of associative and commutative properties of joins. A join operator can use either the nested loop or sort-merge implementation. Each scan node can use either index scan (using a clustered or non-clustered index) or sequential scan. Finally, predicates are evaluated as early as possible. SYSTEM-R-OPTIMIZER uses cost model and The cost model assigns an estimated cost to any partial or complete plan in the search space. It also determines the estimated size of the data stream for output of every operator in the plan. It relies on (a) A set of statistics maintained on relations and indexes, e.g., number of data pages in a relation, number of pages in an index, number of distinct values in a column (b) Formulas to select estimatory of predicates and to project, the size of the output data stream for every operator node. For example, the size of the output of a join is estimated by taking the product of the sizes of the two relations and then applying the joint selectivity of all applicable predicates. (c) Formulas to estimate the CPU and I/O costs of query execution for every operator. These formulas take into account the statistical properties of its input data streams, existing access methods over the input data streams, and any available order on the data stream (e.g., if a data stream is ordered, then the cost of a sort-merge join on that stream may be significantly reduced). In addition, it is also checked if the output data stream will have any order.

The cost model uses (a)-(c) to compute and associate the following information in a bottom-up fashion for operators in a plan: (1) The size of the data stream represented by the output of the operator node. (2) Any ordering of tuples created or sustained by the output data stream of the operator node. (3) Estimated execution cost for the operator cost model assumes that in order to obtain an optimal plan for a SPJ query Q consisting of k joins, it suffices to consider only the optimal plans for sub expressions of Q that consist of (k-1) joins and extend those plans with an additional join. In other words, the suboptimal plans for sub expressions of Q (also called subqueries) consisting of (k-1) joins do not need to be considered further in determining the optimal plan for Q. Accordingly, the dynamic programming based enumeration views a SPJ query Q as a set of relations \{R1,..Rn\} to be joined. The enumeration algorithm proceeds bottom-up. At the end of the j-th step, the algorithm produces the optimal plans for all subqueries of size j. To obtain an optimal plan for a subquery consisting of (j+1) relations, we consider all possible ways of constructing a plan for the subquery by extending the plans constructed in the jth step. For example, the optimal plan for \{R1,R2,R3,R4\} is obtained by picking the plan with the cheapest cost from among optimal plans for (1) Join(\{R1,R2,R3\},R4) (2) Join(\{R1,R2,R4\},R3) (3) Join (\{R1,R3,R4\},R2) (4)Join(\{R2,R3\},R4), \{R1\}. The second important aspect of System R optimizer is the consideration of interesting orders. Let us consider a query that represents the join among \{R1,R2,R3\} with the predicates R1.a = R2.a = R3.a and that the cost of the plans for subquery \{R1,R2\} are x and y for nested-loop and sort-merge join respectively and x y. In such a case, while considering the plan for \{R1, R2, R3\}, we do not consider the plan where R1 and R2 are joined using sort-merge. t if sort-merge is used to join R1 and R2, the result of the join is sorted on a. The sorted order may significantly reduce the cost of the join with R3. Thus, pruning the plan that represents the sort merge join between R1 and R2 can result in sub-optimality of the global plan. The problem arises because the result of the sort merge join between R1 and R2 has an ordering of tuples in the output stream that is useful in the subsequent join but the nested-loop join does not have such ordering. Therefore, given a query, System R identified ordering of tuples that are potentially consequential to execution plans for the query. In the System R optimizer, two plans are compared only if they represent the same expression as well as have the same interesting order. Any characteristic of a plan that is not shared by all plans for the same logical expression, but can impact the cost of subsequent operations. Finally, the System-R’s approach considers physical properties demonstrates a simple mechanism to handle any violation of the principle of optimality, not arising only from physical properties.

3. WEB-BASED DATA INTEGRATION

Allowing uniform querying of information sources has been a major goal of several research efforts. Most proposed systems and techniques focused on making such uniform querying feasible despite all types of hurdles (e.g., heterogeneity, autonomy, unpredictability of the Web, etc.). Achieving the full potential of uniformly querying disparate Web information sources is fundamentally dependent on devising adequate query optimization techniques. Different approaches have been used for Web-based data integration. Most of the systems and techniques in this survey fall into that category. There are also other approaches based on the use of agents ontology and information retrieval techniques [2,26,38].

3.1. Mediator based approaches
Architecture of mediator systems is shown in below figure. Mediators provide an integrated view or mediated schema over multiple heterogeneous and autonomous information sources.

This schema represents generally a synthesized view over a specific application domain. Users access the integrated view through a uniform interface that offers location, model, and interface transparency. In general, each source is connected to a wrapper that enables its participation in the system. It translates between the source’s local language, model, and concepts and those at the mediator level. To resolve a query, a mediator typically performs three main tasks [17]:

- **Database selection.** Locate and select the databases that are relevant to the query.
- **Query translation.** Decompose the query into sub-queries with respect to the previously selected databases. Each sub-query is transformed into a form that is executable by the corresponding database. The sub-query is then sent to the database (through a wrapper) and results are retrieved.
- **Result merging.** Combine the different results into a global answer to the user.

An important characterization of mediator systems relates to the nature of the relationship between the mediated schema and the schemas of participating databases. Two main approaches adapted [20] are:

- In the **source-centric approach**, relations stored by the information sources are described in terms of the global schema. In the **query-centric approach**, the global schema is defined in terms of the relations stored by the local information sources. The source-centric approach performs better than the query-centric since modifications at the information sources do not affect the rest of the system. Translating global queries into local ones is easier in the query-centric approach.

### 3.2. Research issues

Query optimization is essential in different types of database systems (e.g., central, distributed, and multidaybase). The ultimate goal of any database system is to allow efficient querying. Query optimization is base for the deployment of data integration systems over the Web. This is challenging task note to the dynamic and heterogeneous nature of the web. Queries over Web information sources may be answered in various ways and each alternative delivers same results and may differ in terms of efficiency. This may relate to response time, network resources, number of information sources involved, quality of the information being accessed, quality of returned results, users’ satisfaction, and so on. Query optimization techniques for the Web involves Devising the right techniques to address the prevailing issues. We outline issues that are directly related to query optimization over data integration systems on the Web.

**Optimization paradigm.** Optimizing queries results in minimizing response time of the queries through performance tuning. Key features to be considered while optimizing a query are: Optimizing over a large number of heterogeneous and autonomous information sources. Data integration faces a far more incoherent environment than in the pre-Web era. Heterogeneity can exist at different levels of the data integration system. The time and resources required to bridge that heterogeneity may have an important impact on the optimization process. Autonomy, has a more serious impact since several optimization techniques require specific information from information sources. Furthermore, once a (sub-)query is submitted to a specific information source, the optimizer of the data integration system does not have any control over it. Finally, the Web is witnessing an exponential growth in terms of information sources and potential users. Query optimization should consider scalability issues to avoid performance degradation. This degradation could lead to very inefficient query execution plans.

**Evolving in a dynamic environment.**

A major characteristic of the Web is dynamism and volatility. Information sources availability and behavior can change and unpredictable events could happen anytime during query processing and execution. The query optimizer would need adaptive mechanisms to avoid missing optimal query execution in the occurrence of any event. Adaptive techniques could be also used to gather optimization information and use them to modify the execution plan.

**Dealing with limited query capabilities.** Web information sources do not exhibit the same capabilities in terms of the queries they can handle. These limitations are due to several reasons including performance and security. They may range from fully featured database systems allowing full access to their content to information sources with restricted access forms. The query optimizer must generate efficient query execution
plans that are effectively executable with respect to any constraint imposed by the accessed sources. This may especially mean that some optimal plans are skipped since they are not feasible.

Locating sources of interest. Queries do not generally specify the specific information sources that need to be accessed. In some cases, finding those sources is a straightforward process, e.g., query unfolding in global-as-view based mediators. While in others it is more challenging. More precisely, the query optimizer should be able to limit access to only:

(1) relevant sources to avoid wasting resources, and
(2) sources with superior qualities if alternative sources compete in offering the same information. Addressing this issue is dependent on how much meta-information is available about information sources.

3.3. Dimensions for query optimization on the Web

In Web data integration systems, there is a lack of a comparison of the different approaches. This is due to the lack of an analytical model to compare query optimization techniques. In classical systems, well established models have been developed for such purpose. The following proposed dimensions need to be considered for Query optimization on web.

• Scalability. This dimension measures the degradation, if any, of the query optimization technique, when the number of information sources grows. Growth could be in terms of information sources and users.

• Autonomy preservation. This relates to how much information is required from the information source for deploying the optimization technique.

• Optimization performance. This dimension should provide performance evaluation of the objective function under different scenarios. Scenarios may be varied based on the types of queries being submitted, the number and types of sources being accessed, and other parameters.

• Adaptiveness. This reflects the ability of the optimization technique to consider unexpected changes.

• Capabilities. This dimension is for checking whether the optimization technique takes into account sources with different and limited query capabilities. There is a need also to check that the technique does not miss optimal plans.

4. Optimization Techniques

(i). Cost based optimization

Cost incurred while generating information and implementing Query optimization on web based data integration techniques. Below mentioned approaches consider the statistics computed for the submitted query.

(a) Disco. Disco [48] is a mediator system based on the global-as-view approach. The mediator generates multiple access plans involving local operations at the information sources and global operations at the mediator level. The data model is based on the ODMG standard. The object definition language is extended to allow multiple extents to be associated with an interface type of the mediator. It also provides type mapping information between a mediator and a data source. Disco uses a cost based optimization approach [32] and it combines a generic cost model with specific cost information exported by wrappers. The generic cost model uses cost formulas established by the calibrating approach and the data source interface is specified using a subset of CORBA IDL extended with a cardinality section for data source statistics and a cost formula section for specific formulas. The wrapper writer exports statistics (e.g., cardinality of a collection), size rules (reflect the change in result sizes due to an operation), and cost computation rules (compute cost estimates). A query is optimized by estimating the cost of single operations and entire plans. The mediator selects the most specific information available from wrappers and the generic cost model. The cost of the execution of a query plan is determined in a two step bottom-up algorithm:

cost formula integration and cost estimation. In the first step, wrapper rules are integrated into the mediator cost model. In the second step, cost estimates for a plan are generated. The plan is represented as a tree of operator nodes. It is traversed from the root to the leaves and then from the leaves to the root. In the first traversal, cost formulas are associated with nodes. In the second traversal, the cost of each operator is computed.

(b). Garlic. Garlic [21,41] provides an integrated view over heterogeneous information sources using the global-as-view approach as shown in below figure. Query processing is based on dynamically determining the middleware and wrapper roles in answering a query. A wrapper provides several tasks including: (1) modeling information sources as Garlic objects, (2) allowing Garlic to retrieve references to these object, (3) allowing Garlic to invoke methods on objects and retrieve attributes, (4) participating in query processing and execution. Garlic objects have an interface that abstractly describes the object’s behavior and a corresponding implementation that provides a concrete realization. A query is translated into a tree of operators or POPs (Plan Operator’s). Each operator corresponds to a runtime operator, e.g., join, sort, fetch, and scan. Garlic provides also a generic POP, called Pushdown POP, which encapsulates work to be conducted at a information source. Each plan is characterized by properties like tables used in the plan, output columns, and estimated cost. Garlic extends the traditional cost-based optimization approach by involving wrappers as important components in the
optimization process. Wrappers cooperate in the estimation of the total cost of a given query plan. The proposed framework aims to provide the necessary means to extend the traditional cost-based optimization to a heterogeneous environment [40]. The framework includes a cost model, cost formulas, and statistics. Wrappers may use a default cost model or a more specific one to model their execution strategies. The default cost model considers the total cost of a POP operator as a combination of the cost of two basic tasks: reset for initializing the operator and advance for retrieving the next result. Default cost formulas are used to compute these two costs based on statistics. Based on the cost of the reset and advance tasks, the wrapper should be able to provide cost estimates for its own plans that are included in the global query plan. The total cost, re-execution cost, and result cardinality are considered in the cost of a Pushdown POP. They are required to estimate the cost of the global query plan. Once the optimizer selects a winning plan, that plan is translated into an executable form. Garlic POPs are translated into operators that can be directly executed by the Garlic execution engine.

(ii) Quality-based optimization techniques

(a). Object Globe. Object Globe’s [7] data integration is centered around three types of suppliers: data suppliers, function providers for query processing operators, and cycle providers for operators execution. The execution of a query may involve query operators supplied by different function providers that are executed at different cycle providers and that retrieve data from different data providers. Query processing follows a multi-step strategy. First, a lookup service locates instances from each of the above types of suppliers that are relevant to the resolution of the query. Some cost information are also gathered during this step. In the second step, an optimal plan is built based on a cost model using the previously gathered information. In the last step, the query execution plan is distributed to relevant cycle providers and executed using an iterator model [26]. Queries for the lookup service are extracted by the parser based on themes and attributes specified in the different clauses of a query. The lookup service uses the generated queries to locate relevant resources by consulting a meta-data repository. The lookup service is also responsible to gather statistical information for the optimizer and authorization information from the different providers. Authorization information is recorded in a compatibility matrix that will annotate the query execution plan. The optimizer enumerates alternative query execution plans using a System-R like dynamic algorithm. Costs of plans are estimated using information from the lookup service. If an information is missing it is set to a default value. The optimal plan is then executed by distributing it to the different cycle providers as specified in the host annotations of the plan. In addition, users can specify quality constraints on the execution of their query. Constraints are defined on results (e.g., size of the result), cost (i.e., how much the user is ready to pay), and time (e.g., time to first results). QoS management is introduced as part of the query processor. If the quality constraints are not fulfilled, the query plan is dynamically adapted or the query is aborted. Based on that QoS concept, the optimizer’s goal is to maximize the percentage of successful queries and abort any query that cannot fulfill its QoS constraints as soon as possible.

(b). HiIQIQ. HiIQIQ (High Quality Information Querying) uses information quality criteria to support query optimization in mediator systems [34]. The focus is on incorporating information quality into query planning. This is useful in environments like biological information systems where users are more sensitive to quality criteria than the classical database criterion, i.e., response time. The approach uses query correspondence assertions (QCAs) to resolve queries. QCAs are set oriented equations involving queries over mediator and wrapper schemas. This is a sort of combination of the source-centric and query-centric approaches used in mediators. The query resolution process tries to find all correct plans (combinations of QCAs) and then does a union over their results to obtain the complete answer. To include quality information (IQ) in the query planning, three classes of quality criteria have been used. Source-specific criteria determine the overall quality of a source. They include ease of understanding, reputation, reliability, and timeliness. QCA-specific criteria determine quality aspects of specific queries that are computable by a source. These include availability, price, response time, accuracy, relevancy and representational consistency. Attribute-specific criteria relate to the ability of a source to provide the attributes of a specific query. Query processing is conducted in a three phase strategy. Based on the source specific IQ criteria, the first phase prunes low-quality sources. The second phase finds all plans, i.e., combinations of QCAs, that produce semantically correct answers. Finally, plans obtained from the previous phase are qualitatively ranked. This phase starts by determining IQ scores for the different QCAs in the different plans. That score is a vector with eight dimensions; each dimension corresponding to a non-source-specific criteria (six QCA-specific and two attribute-specific). Only attribute-specific criteria are recalculated for
each new query, the others are more stable. In a second step, an IQ vector is computed for each plan by merging IQ vectors in join nodes appearing in the plan. In the third and last step, IQ scores for the different plans are scaled, weighted, and compared using a simple decision making method (i.e., the simple additive weighting). This method scales the scores to make them comparable, apply the user weighting, and sum up the scores for each plan. 

(iii). Adaptive query optimization

An important issue in Web-based data integration is the use of adaptive or dynamic query optimization techniques to face the high volatility of the Web. These techniques address mainly the lack of statistical information about the data sources. Both relations are described using a third type, schematics relations. Relations resulting from wrappers are used by users to formulate queries. Second, the structure of the information sources. Both relations are described using a third type, world relations, that represents a reference schema for the system.
Query optimization is based on the notion of **local completeness** that limits the number of sources to be queried. Info master uses a weak form of completeness that assumes that some subset of the data stored by an information source is complete. This is assumed even if the entire data stored by this information source might not be complete. The claim is that such assumption is reasonable within a specific application domain. In this approach, any source relation is represented by two views over the global relations. The **liberal view** and **conservative view**. These two views have the same schema as the corresponding source relation. The conservative view is a lower bound of the source relation and describes the subset of the data that is known to be complete. The liberal view is an upper bound of the source relation. Based on conservative and liberal views of the different information sources, the query processor generates a query plan that is: (1) retrievable, (2) semantically correct, (3) source complete, and (4) view minimal. The query processor generates such plans in a five step algorithm.

(1) **Reduction**: The query is transformed in terms of base relations by rewriting each atom using definition of the associated predicates.

(2) **Abduction**: Given the expression $Q_b$ in terms of base relations and a set of definitions, the set of all consistent and minimal conjunctions and retrievable atoms is produced. This set is contained in $Q_b$ with respect to the given definition.

(3) **Conjunctive minimization**: Any redundant conjunct is eliminated.

(4) **Disjunctive minimization**: Any disjunct that can be shown to be contained in the union of the remaining disjuncts is discarded.

(5) **Grouping**: The conjuncts within each conjunction are grouped so that the atoms in each group share a common provider.

---

[d] **SUTS**: SUTS [51] is a search interface which provides a layer of abstraction on top of relational databases to smoothly integrate the flexibility of keyword search and the precision of database queries and performs Query optimization. The architecture of SUTS is shown in the figure below. Processing steps can be split into two phases: an offline pre-computing phase and an online query phase. In the pre-computing phase, SUTS creates inverted indices for all text columns in the database, which will be used in both query generation and query execution. It also generates query templates that are potentially employed by users when forming structured queries. For example, users sometimes search for movies with a certain character, and sometimes for actors who played in a certain movie. These are all meaningful query templates to be generated in the pre-computing phase. The online query phase consists of three steps. In Step 1, the system receives the user’s keyword query and passes it to the full text indices to check for occurrences of the query terms in all tables and attributes. In Step 2, it combines these term occurrences with the pre-computed query templates to generate meaningful structured queries. In Step 3, the system ranks the structured queries according to their likelihood of matching the user’s intent and returns the top-k queries with non-empty result-sets. When generating structured queries in Step 2, the system also generates query construction options that the users can use later in incrementally refining their keyword queries. These options are ranked in Step 3 and returned to the user. If the user selects some of these options, the selected options are passed together with the keywords to the query generation step (Step 2) to filter out queries that do not satisfy the selected options.
The best way to tune performance is to try to write queries in a number of different ways and compare their reads and execution plans. Various techniques can be used to optimize database queries in order to improve the performance of SQL queries [33]. Developers need to understand the query optimizer and the techniques it uses to select an access path and prepare a query execution plan. Query tuning involves knowledge of techniques such as cost-based and heuristic-based optimizers, plus the tools an SQL platform provides for explaining a query execution plan.

**Query Performance using IO Statistics:**

There are different ways to write the best way to write queries. Two of common methods are looking at the number of logical reads [46] produced by the query and looking at graphical execution plans provided by SQL. For determining the number of logical reads, you can turn the STATISTICS IO option ON. Consider this query:

```sql
SELECT * FROM table_name
```

The following is returned in the Messages window in SQL tool:

```
Table _table_name_. Scan count 1, logical reads 33, physical reads 0, lob reads 0, lob physical reads 0, lob read-ahead reads 0. There are several bits of data returned by STATISTICS IO, but the logical reads portion plays a vital role because it retrieves the number of pages read from the data cache. This is the most helpful because it will stay constant when we run the same query, which is important because there are external factors that might vary the execution time of my queries, like locking by other queries. When tuning SQL queries, goal should be to get the number of logical reads as low as possible because fewer logical reads typically lead to faster execution times.

**GENERAL TIPS FOR QUERY OPTIMIZATION:**

1. **Specific Column Names instead of * in SELECT Query**

   The SQL query becomes faster if you use the actual column names [46] instead of *. So we need to restrict the queries result set by selecting only the particular columns from the table, rather than all columns from a particular table. This results in performance benefits, as SQL will return only particular columns to the client, not all columns of a table. This will help reduce the network traffic and also boost the overall performance of the query.

   **Example:** Write the query as

   ```sql
   SELECT col_1, col_2, col_3, col_4, subject FROM table_name;
   ```

   Instead of:

   ```sql
   SELECT * FROM table_name;
   ```

2. **Alternatives of COUNT(*) for returning total tables row count**

   If we need to return the table’s row count, we can use alternative ways instead of the SELECT COUNT(*) statement. As SELECT COUNT(*) statement makes a full table scan to return the table’s row count, it can take much time for the large tables. There is another way to determine the total row count of a table. We can use sys indexes system table. There is a ROWS column in the sys indexes table. This ROWS column contains the total row count for each table in a particular database. So, we can use the following select statement instead of —

   ```sql
   SELECT COUNT(*): SELECT table_name FROM sysindexes WHERE id = OBJECT_ID ('table_name') AND indid < 2.
   ```

   So we can improve the speed of such queries thus resulting in better performance.

3. **Try to avoid HAVING Clause in Select statements**

   HAVING clause is used to filter the rows after all columns to the client, not all columns of a table. This will help reduce the network traffic and also boost the overall performance of the query.

   **Example:**

   ```sql
   SELECT Col_1, count (Col_1) FROM table_name GROUP BY Col_1 HAVING Col_1 != 'testvalue1' AND Col_1 != 'testvalue1' GROUP BY Col_1 HAVING Col_1 != 'testvalue1' AND Col_1 != 'testvalue2';
   ```

4. **Minimize number of Sub query blocks within query**

   We may have more than one sub query block in our main query. We should try to minimize the number of sub query block in our query.

   **Example:**

   ```sql
   SELECT col_1 FROM table_name1 WHERE (col_2, col_3) = (SELECT MAX (col_2), MAX (col_3) FROM table_name2) AND col_4 = _testvalue1;
   ```

   Instead of:

   ```sql
   SELECT col_1 FROM table_name1 WHERE col_2 = (SELECT MAX (col_2) FROM table_name2) AND col_3 = (SELECT MAX (col_3) FROM table_name2) AND col_4 = _testvalue1;
   ```

**More Specific tips for Query Optimization**

1. **Use table variables instead of temporary tables as table variables require less locking resources as well as less logging resources than the temporary tables, so table variables should be used whenever possible.**
2. We should try to avoid the use of DISTINCT clause, where ever possible. As the DISTINCT clause will result in performance degradation, we should use this clause only when it is necessary or unavoidable.

3. We need to use TOP keyword or the SET ROWCOUNT statement in the select statements, if we need to return only the first n rows. This can improve performance of our queries, as the smaller result set will be returned. It can also reduce the traffic between the server and the clients.

4. We should Try to use user-defined functions to keep the encapsulated code for reuse in future.

5. Try to use stored procedures instead of heavy queries as they can reduce network traffic. Stored procedures can be used to enhance security. For example, we can give different users, different set of permissions to execute the stored procedure to work with the restricted set of the columns and data.

6. We should try to return an integer value from a RETURN statement instead of returning an integer value as a part of a record set. As the RETURN statement exits from a stored procedure unconditionally, so the statements following the RETURN statement are not executed. The RETURN statement is generally used for error checking, but we can also use this statement to return an integer value for any other reason. Using RETURN statement can improve performance as SQL Server will not create a record set.

7. We should try to drop indexes that are not being used. Because each index takes up disk space and slows the DML operations, we should drop indexes that are not used. We can use Index Wizard to identify indexes that are not being used in our SQL queries.

CONCLUSION AND FUTURE SCOPE:
In this Survey, we presented several techniques providing efficient access to interconnected databases and other information sources to handle large amounts of data which have impact on data integration and Query processing and optimization over the web. Further, this can be extended to build a Query interface where web services are treated as first class objects where in users and applications would submit queries that are resolved by combining invocations to web service operations. there is a need to build an infrastructure to organize and efficiently query Web services where they would be treated as first-class objects and devising a computational model for interaction between web service registries and web portals and successfully deploying them considering performance tuning techniques there by improving Quality of Service(QOS) which is goal of Query Optimization. The techniques described in this paper allow basic optimization of queries, tables, indexes and stored procedures for performance gains.

REFERENCES


[30]. Minyar Sass [MA05], and Amel Grissa-Touzi “Contribution to the Query Optimization in the Object-Oriented Databases” World Academy of Science, Engineering and Technology 11 2005


[38]. Pitchaimalai, S., Ordonez, C., Garcia-Alvarado, C., Efficient Distance computation Using SQL Queries and UDFs IEEE HPDM (High Performance Data Mining Workshop, at ICDM), 2008.


[51]. Xuan Zhou Gideon Zenz Elena Demidova .SUITES: “Constructing Structured Queries from Keywords” Wolfgang NejdlL3S Research Center & Leibniz University of Hanover 30167, Germany January 2012.
