QUALITY-ADAPTIVE QUERY PROCESSING OVER DISTRIBUTED SOURCES

(Research-in-Progress)

Laure Berti-Équille
IRISA, University of Rennes I, France
Laure.Berti-Equille@irisa.fr

Abstract: For non-collaborative data sources, both cost estimate-based optimization and quality-driven query processing are difficult to achieve because the sources do not export cost information nor data quality indicators. In this paper, we first propose an expressive query language extension using QML[1] syntax for defining in a flexible way dimensions, metrics of data quality and data source quality. We present a new framework for adaptive query processing on quality-extended query declarations. This processing includes the negotiation of quality contracts between the distributed data sources. The principle is to find dynamically the best trade-off between the cost of the query and the quality of the result retrieved from several distributed sources.

Key Words: Data Quality, Source Quality, QML, Quality of Service, Negotiation, Adaptive Query Processing.

1. INTRODUCTION

For the mediator/wrapper architecture, the access to distributed information sources is carried out in a declarative way. The mediator processes the queries of the users at the global level and optimizes the query plans according to the wrappers that reach respectively their underlying data sources. In this type of environment, the sources are usually non-collaborative and do not export information describing the local costs of query processing, neither indicators of their quality of service (e.g., resource accessibility, reliability, security, etc.) nor information describing the quality of their content (e.g., data accuracy, freshness, completeness, etc.). Not provided, this information has to be estimated, computed dynamically and updated if provided. But, the lack of scalability and flexibility face to the growing number and the changing structure of data sources harm seriously the effectiveness of dynamic execution of distributed queries and the quality of results. Although there are several approaches that deal with the assessment and the management of quality metadata, the dual problem of fixing the query cost and optimizing the result quality, or fixing the result quality and optimizing the query cost still remains. Querying simultaneously several data sources in a dynamic and distributed environment raises several interesting problems and open issues as follows:

- Selecting dynamically appropriate sources: different information sources may answer the global query with different response times, query costs and various levels of result quality. How to define strategies for selecting adaptively the most appropriate sources for answering the whole or parts of the global query with the right quality? Which are the criteria to select the best quality sources on the fly?

- Defining semantically and qualitatively correct distributed query plans: the result of a global

[1] Quality of Service Modeling Language proposed by Frølund and Koistinen [9]

query is built according to the particular order for execution of subquery plans. This must combine in a coherent way both information and meta-information from the various sources; but data quality levels are often unknown, heterogeneous from one source to another (inter-source quality heterogeneity) and locally non uniform (intra-source quality non-uniformity): in this context, the aim is to control and merge data quality indicators in a consistent way for both correctly integrating data and quality metadata

- Making trade-offs between the query cost and the final and perceived result quality (including both the quality of service and the quality of content): because we may accept a query result of lower quality (if it is cheaper or has a shorter response time than if the query cost is higher), it's necessary to adapt query costs to users' quality requirements. The objective is to measure and optimally reduce the cost and bargain query situations where the system searches for solutions that "squeeze out" more gains (in terms of result quality) than the query without quality requisites.

- Developing concrete cost models to evaluate whether the expected benefits from the improved query plan and decision quality compensate for the cost of collecting and evaluating feedback from the environment during execution time. The difficulty is to adapt existing query processing techniques to environments where resource availability, allocation, quality and cost are not, by definition, decidable at compile time.

For these open questions, some solutions have been proposed in the literature, but very few approaches use the quality of service approach together with the quality of data techniques for adapting query processing. In this context, our objectives are: (i) to extend a generic query language for describing and manipulating, in a flexible way, data and source quality contracts, (ii) to propose a new query processing framework for selecting dynamically sources with quality-extended queries with a negotiation strategy.

This paper describes *XQuaL - QML-extended Query Language* - and the negotiation processing over distributed sources with heterogeneous quality. While our general approach to query processing is typical, a number of factors associated with quality constraints specification and negotiation complicate the problem of distributed query processing. The challenges have been to choose an appropriate formalism for specifying, in a flexible way, quality metadata (dimensions and metrics) and to devise methods for quality-extended query processing including a quality-based negotiation algorithm.

The paper is organized as follows. In Section 2, we briefly present previous work in the areas of quality of service (*QoS*) and of quality of Data (*QoD*). In Section 3, we describe the specifications of our query language extension. In Section 4, we present the definitions and the algorithm for the negotiation for the quality-extended query processing. In Section 5, we present an illustrative example. Section 6 concludes the paper and presents our future work.

2. RELATED WORK: WHEN QOS MEETS QOD

2.1. Quality of Service

Quality of service (*QoS*) has gained much attention in the field of distributed multimedia applications where it covers essentially the notion of end-to-end guarantees associated with the communication of multimedia contents over networks and their processing on computing nodes. A considerable amount of work has been done in this area to introduce models and architectures supporting such QoS guarantees. This narrow vision of QoS (generally centered on the concept of performance) is currently generalized to extra-functional properties of distributed systems [10][1]. The traditional acceptance of quality of service for distributed information systems has been recently revised with a broader

definition raising new problems. Traditional quality of service categories and dimensions (such as execution time or cost) are now extended to integrate the concepts of data quality and source quality as well as quality of provided services (such as searching techniques, optimization strategies, data transfer, adaptation or presentation). In this context, the QML language (*Quality of service Modeling Language*) has been proposed by Frølund and Koistinen [9] in order to deal with the quality of service of software components in a systematic and declarative manner, allowing flexible definition of quality dimensions, quality contracts and profile constraints. We found relevant to adapt the language QML for extending SQL-like query language syntax and querying data with quality requirements.

2.2. Quality of Data

Literature on data quality proposes definitions, metrics, conceptual models and methodologies to improve data quality in databases (see Dasu and Johnson [6,7]), information systems (see Wang [37,36,32], Ballou and Pazer [2], Pipino and Lee [25], Fox, Letivin and Redman [8], [27]) and data warehouse systems (e.g., *DWQ - Data Warehouse Quality*, see Clavanese et al., [4], Vassiliadis [34,35]). Most of the approaches are centered on various methods of imputation such as inferring missing data from statistical patterns of available data sets, predicting accuracy of the estimates based on the given data, data edits (automatic detection and handling of outliers in data), error control (see Paradice and Fuerst [24]). Various methods were developed to measure the data quality provided to the users in conformance to their quality specifications or by comparison with a referential database. The use of metadata for evaluation and improvement of data quality was also recommended by Rothenberg [28] where information producers were encouraged to carry out the verification, the validation, and the certification (*VV&C*) of their data.

The most frequently mentioned data quality dimensions in the literature are accuracy, completeness, freshness and consistency:

- Correctness is measured by detecting the rate of incorrect values in the database (see Hou, Zhang, [15]),
- Completeness is measured by detecting the rate of missing values in the database,
- Freshness is measured by detecting the rate of obsolete values in the database,
- Consistency is measured by comparison with a set of constraints by detecting the data, which do not satisfy them.

But many others dimensions, metrics and measurement techniques have been proposed in the literature (by Fox, Letivin, Redman, [8][27], Naumann [21,22], by Kahn, Strong, Wang [16], by Ballou and Pazer [2]). The general trend is the use of Artificial Intelligence methods (training, knowledge representation schemes, management of uncertainty, etc.) for purposes of data validation (see Maletic, Marcus [19], Lee et al. [18], Fayyad, Piatetshy-Shapiro [11], Winkler [38], Dasu and Johnson [6,7]. The use of machine learning techniques for data validation and correction was first presented by Parsaye, Chignell [23]: rules inferred from the database instances by machine learning methods were used to identify outliers in data and facilitate data validation process. Another similar approach was proposed by Schlimmer [30,31]. Utilization of statistical techniques for improving the correctness of the databases and introduction of a new kind of statistical integrity constraints were also proposed by Hou and Zhang [15]. Statistical constraints are derived from the database instances using conventional statistical techniques (sampling, regression).

Other propositions concern the definition of declarative language extensions for specifying/querying quality metadata (*Q-Data* by Sheth, Wood, Kashyap [33]) or for applying data transformations necessary to specific cleaning process (*AJAX* by Galhardas et al. [12], ARKTOS [35], Potter's Wheel [26]). The prototype described of Sheth, Wood, Kashyap [33] checks if the existing data are correct: the authors call it data validation and cleanup by using a logical database language (*LDL++*). The system employs data validation constraints and data cleanup rules. *AJAX* [12] is an SQL extension for specifying each data transformation (such as matching, merging, mapping, clustering) for the data

cleaning process. These transformations standardize data formats when possible and find pairs of records that most probably refer to the same object. The duplicate-elimination step is applied if approximate duplicate records are found and multi-table matching computes a similarity join between distinct data flows and consolidates them.

Among the projects that have been proposed for data quality in distributed environments: *HiQ B&B* (*High Quality Branch and Bound Algorithm*) proposed by Naumann [21][22] is a distributed query-planning algorithm that enumerates plans in such way that it usually finds the best *N* plans after computing only a fraction of the total number of plans. Upper quality bounds for partial query plans are constructed and thereby non-promising subplans are early pruned in the search tree. This integrates the source and plan quality-based selection phases to a planning algorithm that finds the top *N* plans for a query distributed over several quality-heterogeneous sources.

2.3. Motivations

Positioned between quality of data and quality of service, our approach extends a declarative query language for taking into account, in a flexible way, aspects and constraints related to data source quality. We also adapt the query processing in order to find the trade-off between the query cost and the quality of result. Our query processing framework takes as input a quality-extended query and attempts to adapt the processing to:

1. **User preferences and quality requirements**. Adapting to user preferences includes cases where users are interested in obtaining some partial results of the query quickly with specific constraints on data quality and source quality of service. In order to meet such needs, the query processor may, for instance, produce results incrementally, as they become available. The user can also classify the elements of the output in terms of importance or quality requisites and, in that case, the query evaluator has to adapt its behavior in order to produce more or better quality results earlier. Thus, the query processor has to consider potential input from the user, such as priority ratings and quality declarations for different parts of the result.

2. **Data source statistics.** In some cases, it is not possible at compile time to gather accurate statistics about the data sources content and quality. A solution to this problem is to collect such statistical information at runtime, thereby ensuring that they are valid for the current circumstances, and to adapt query execution based on it. Techniques that adopt this policy may change the query plan when the actual statistics have become available.

3. EXTENDING QUERY LANGUAGE WITH QML

The syntax of our quality-extended query language, XQuaL is presented in Appendices and is adapted from Frølund, Koistinen [9] to which we refer the reader for more information. As we will present in the next sections, this language extension is a *Data Manipulation Language* (*DML*) with the SQL-like declaration part and also a *Data Description Language* (*DDL*) with the contract type declaration defined into the Qwith part.

3.1. Quality Metadata Description

A contract is a set of constraints that are defined on different user-specified dimensions. Each contract is an instance of a complex contract type. Figure (1a) proposes examples of contract types declarations for the dimensions called Reliability, Freshness, Completeness, and Credibility and defined as contract types. Figure (1b) shows their corresponding contract instances.
The type of the Reliability contract has four dimensions (noted d1, …, d4 as comments of Figure (1a)), named respectively failureMasking, serverFailure, numberOfFailures and availability. Dimension failureMasking is

defined by the possible values among omission, lostResponse, noExecution, etc. and the relation (decreasing) is interpreted such as the smallest sets of values are the preferable values for this dimension. Dimension serverFailure is defined by individual values named without order. Dimensions numberOfFailures and availability are numerical dimensions. nf/year is the number of failures per year as a unit of numberOfFailures.

Figure (1b) presents four instances of these contract types for the source S1 such as: S1_Reliability, S1_Freshness, S1_Completeness, and S1_Credibility.

```
type Reliability = contract {
        failureMasking : decreasing set {omission, lostResponse,
                                noExecution, response, responseValue, stateTransition } ;   //cl1
        serverFailure : enum { halt, initialState, rolledBack } ;                           //cl2
        numberOfFailures : decreasing numeric nf / year ;                                   //cl3
        availability : increasing numeric ;                                                 //cl4
} ;
type Freshness = contract {
        dataAge : decreasing numeric seconds;                                               //cl5
        lastUpdate : numeric seconds ;                                                      //cl6
        updateFrequency : numeric uf / month ;                                              //cl7
} ;
type Completeness = contract {
        percentageOfNullValues : percentile number %;                                       //cl8
        numberOfNullValuesPerQuery : numeric nnv / query ;                                  //cl9
} ;
type Credibility = contract {
        reputation : enum {veryGood, good, bad, veryBad, unknown } ;                        //cl10
} ;
```

Figure (1a). *Example of contract types*

```
S1_Reliability = Reliability contract {
  failureMasking < { noExecution, response };
  serverFailure == initialState ;
  numberOfFailures < 10 nf / year ;
  availability > 0.8 ;
} ;
S1_Freshness = Freshness contract {
  dataAge < 4200 seconds;
  lastUpdate == 4500 seconds;
  updateFrequency == 3 uf / month;
};
 S1_Completeness = Completeness contract {
    PercentageOfNullValues == 8 %;
    NumberOfNullPerQuery == 2 nnv/query;
 };
 S1_Credibility = Credibility contract {
    reputation == veryGood ;
 };
```

Figure (1b). *Example of contract instances*

A quality-extended query (Qwith-query) is a SQL-like query followed by a Qwith operator used to declare the types of contracts (contractTypeDeclaration), the contracts (contractDeclaration) or the quality profiles (profileDeclaration) – see Appendices –. Quality can be defined in a flexible way as a specific combination of contract types with a relation of order on data and source quality constraints. Figure 2 presents an example for the definition of quality as a complex contract type using the previously defined contract types of Figure (1a).

```
type Quality contract {
 sourceQuality : increasing set {Reliability,Freshness,Completeness,Credibility }
     with order {Reliability<Completeness, Completeness< Freshness, Freshness < Credibility }
 } ;
```

Figure 2. *Example of quality contract declaration*

3.2. Quality Metadata Manipulation

A profile is the set of the required contracts that are affected to one (or several) data source(s) or functionalities. The profiles are useful to associate contracts to sources interfaces (or wrappers).
More precisely, a profile declaration (profileDeclaration) is defined with the identifier of the profile, the identifier of the interface and a series of expressions (profileExpression) requiring the satisfaction of contracts by the entities of the targeted interfaces/wrappers (requisites) – see Appendices–.

The profile declaration makes it possible to associate quality contracts with targeted source interfaces and their specific services or methods (such as query_answer_method for source S1 in Figure 3).

```
Source_Profile for S1 = profile {
        require S1_Freshness, S1_Completeness, S1_Credibility ;
        from S1.query_answer_method require S1_Reliability contract ;
} ;
```

Figure 3. *Example of source profile for Source S1*

The clause require binds all the interface/wrapper entities to a list of contracts, whereas the clause from... require... specifically identifies a list of interface entities to which the list of contracts definitions will be associated in the following clause require.

Our contribution is to include the principle of quality contract negotiation over distributed sources and to adapt the query processing. Our objectives are to incorporate a set of primitives and data quality functions that extend the query and that can be computed at runtime and be used for adapting the query processing. Our approach includes: (i) specification of quality requirements or capabilities in the declarative language XQuaL, (ii) negotiation: reaching an agreed specification between all parties (e.g., between the mediator and the wrappers or between the integration system and the sources), (iii) admission control: comparison between the required quality and the capability to meet them, (iv) monitoring: measuring the quality actually provided by the distributed sources, (v) policing: ensuring that all parties adhere to quality contracts, (vi) maintenance: modification of the system parameters to maintain the quality, and (vii) renegotiation: modification of the quality contracts and requisites.

4. NEGOTIATION FOR QUALITY-EXTENDED QUERY PROCESSING

4.1. Definitions

Negotiation can range over number of quantitative and qualitative dimensions of quality. Each successful negotiation requires a range of such dimensions to be resolved to the satisfaction of both parties (i.e. the querying and the queried systems). Making trade-offs between query costs and result quality is required in order to come to an agreement for "good quality" query results. The structure of negotiation is based almost directly on the quality contract used to regulate agreements, which is fairly rich and may cover content quality, service quality and meta-service attributes. Let us now outline the formal basics of our quality-extended query negotiation model.

Let i ($i \in [1..n]$) representing the data sources and j ($j \in [1..k]$) be the dimensions of quality contracts under negotiation (e.g., dataAge, failureMasking, serverFailure, availability numberOfNullValuesPerQuery, etc.). Let $x_{ij} \in [min_{ij}, max_{ij}]$ be a value for the quality dimension j that is acceptable to the user for querying the source i.

At the given time t, each source has a scoring function $score_{ij}(t)$: $[min_{ij}, max_{ij}] \rightarrow [0,1]$ that gives the score of the source i assigned to a value of the dimension j in the range of its acceptable values. For convenience, scores are kept in the interval [0,1]. The relative importance that the user assigns to each dimension under negotiation is modeled as a weight, noted w_{ij}, that gives the importance of the dimension j for the source i. We assume the weights are normalized:

$$\sum_{1 \le j \le k} w_{ij} = 1 \quad \forall i \in [1..n], \forall j \in [1..k] \tag{1}$$

An aggregate scoring function of the source j in the multidimensional space of quality dimensions defined by $x = (x_1,\ldots,x_k)$ is defined as :

$$Score_i(x,t) = \sum_{1 \le j \le k} w_{ij}.score_{ij}(x_j,t) \tag{2}$$

For analytical purposes, we restrict our study to an additive and monotonically increasing or decreasing value scoring function.

Negotiation consists in finding trade-offs between quality profile requisites and effective quality contracts instances of sources during the query processing. The negotiator module decides to make a trade-off action when it does not wish to decrease the aspirational quality level (denoted θ) for a given quality-extended query. The aspirational quality level means the initially required quality. Thus, the negotiator module first computes the quality differential between the effective quality of the sources and the required quality represented by the curve of aspirational quality level θ. And it generates also the contract that lies on the iso-value curve for θ.

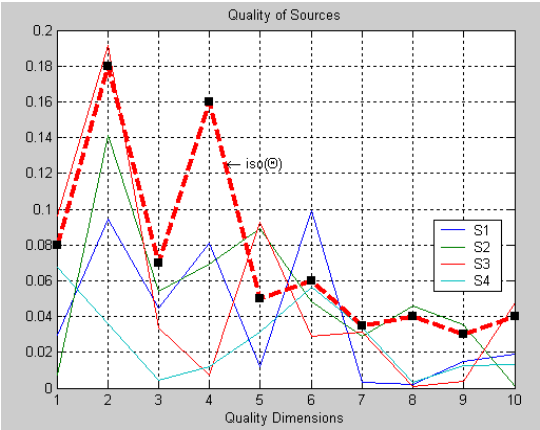

Figure 4a. *Quality for sources S1, S2, S3 and S4*

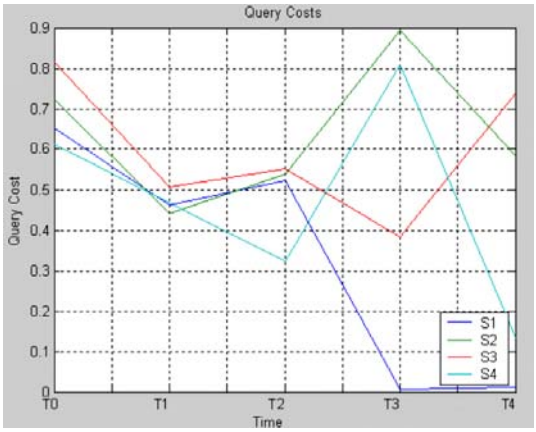

Figure 4b. *Query costs over S1, S2, S3 and S4*

Consider four sources S1, …, S4 and the ten dimensions of quality contracts types previously defined in Figure (1a). Figure (4a) represents the quality dimensions values of each source. The aspirational quality level required initially is represented with the red dotted line. Figure (4b) represents the cost variations of each source in temporal window for a given query. In practice, the query cost of a query q at a given time t is never null, $Cost_i(q,t) \in]0,1]$.

The aim of the trade-off mechanism is to find the contract that is the closest to the initial aspirational curve or the most preferable (i.e, that maximizes the quality and minimizes the query cost). More formally, the aspirational iso-curve and the trade-off are defined as follows:

Definition 1. Aspirational Quality Iso-Curve: Given the required aspirational quality level, noted θ, the iso-curve set at level θ is defined as the set of values of quality dimensions that are the most similar to θ:

$$iso(\theta) = \{x_j \mid \max(Sim(score_{ij}(x_j),\theta_j))\ \forall i \forall j\} \quad (3)$$

The heuristic we employ is to select the contract that is the most "similar" to the negotiator's last proposal (since this may be more acceptable to the user in terms of quality requirements).
The similarity between x and y over the set of quality dimensions D is defined as:

$$Sim(x, y)=\sum_{j\in D} w_j.distance_j(x_j,y_j) \quad \text{with} \sum_{j\in D} w_j=1 \quad (4)$$

We choose $distance_j$ as an Euclidean distance function for dimension j.

Definition 2. Local Trade-Off: Given a query q and its quality extension x, the local trade-off of the source i, at the given time t, for the quality-extended query (q, x) is defined as follows:

$$TradeOff_i(q,x,t)=\frac{Cost_i(q,t)}{1+Score_i(x,t)} \quad (5)$$

4.2. Negotiation Strategy

Intuitively, the principle of our negotiation strategy is as follows: we define a system state as the query execution cost and the quality of the source at a given time. Then, we search for the well-balanced state (i.e., the trade-off) that is the state with the lowest cost and the best quality in conformance with the quality requirements defined into the contracts. Our objective is to find the trade-off state out off in a local minimum state with an adaptation of the simulated annealing process [17]. First, we choose randomly a candidate source among the sources. We evaluate its cost and quality. We examine others query plans over the sources answering the query and put them into the candidate list even if their costs is greater than the best current one. We accept cost increasing when quality also increases for getting out of local minimum and converge to the global minimum of cost for the global maximum of quality. If no trade-off is found, we modify the quality contracts (i.e. we soften the constraints) and restart the process with new costs and new qualities.

```
Negotiation(SourceSet, Query, Aspirational_Quality_Θ )
Initialization(Current_State, T)
                                // Random selection of a current state
                                //Initialization of Temperature T

Calculation Current_State(Current_Cost,Current_Quality_Score)

while T ≠ 0
        while no trade-off
         Calculation New_State(New_Cost,New_Quality_Score)
         Δ cost = Current_Cost - New_Cost
         Δ quality = Current_Quality_Score - New_Quality_Score
         if Δ quality ≥ 0 :
                        if Δ cost ≤ 0 : Current_State = New_State
                        else if e^(-Δcost/T) > Random(0,1)
                                then Current_State = New_State
                                Else no trade-off
                Else
                no trade-off
                Renegotiate quality contract
                                with New_Aspirational_Quality_Θ
        End-while
decrease T
End-while: T ≤ 1 and New_State doesn't change 4 times
```

Figure 5. *Pseudocode for Negotiation*

The algorithm starts from a valid solution and randomly generates new states, for the problem and calculates the associated cost and quality function. Simulation of the annealing process starts at high fictitious temperature (noted T). A new state is randomly chosen and the differences in cost and in quality function are calculated. If Δ Cost $\leq$ 0, i.e., the cost is lower and the quality is monotone or increasing, then this new state is accepted. This forces the system toward a state corresponding to a local or a possibly a global minimum. However, most large optimization problems have many local minima and the optimization algorithm is therefore often trapped in a local minimum. To get out of a local minimum, an increase of the cost function is accepted with a certain probability. For each temperature, the system must reach the equilibrium i.e., a number of new states must be tried before the temperature is reduced typically by 10 %. It can be shown that the algorithm will find, under certain condition, the global minimum and not get stuck in local minima. The source that provides the trade-off is selected for answering the query.

5. A SIMPLE EXAMPLE

Consider a list of queries sent to four overlapping sources with five attributes A1, …, A5 (see the structure of the sources in Table 1). Consider ten quality dimensions (d1, …, d10) previously defined in Figure (1a) for Reliability, Freshness, Completeness and Credibility contract types. Each query (listed in Table 2) is extended with the following quality statement:
query# Qwith quality for Source_Set = profile {require Reliability, Freshness, Completeness, Credibility};
with query# the variable representing the query number in Table 2 and Source_Set={S1,S2,S3,S4}.

The quality of the four sources is given in Figure (4a) over the ten dimensions. Their costs are given in Figure (4b) from the initial time of the query T0.

Sources	Attributes			
S1	A1	A2	A3	A4
S2	A1	A2	A3	A4
S3	A1	A2 < 50		A4
S4	A1	A2 > 40	A3	

Table 1. *Source schema*

Query #	Algebraic Expression	Candidate Sources	Best Source
q1	Select(A1, λa1, true)	S1,S2,S3,S4	s1
q2	Select(A2, λa2, a2>30)	S1,S2,S3,S4	s4
q3	Project(A3, λa3, Join(q1,q2, λa1, a2, (a1=a2)))	S1,S2,S4	S2
q4	Project(A4, λa4, Join(q1,q2, λa1, a2, (a1=a2)))	S1,S2,S3	S1

Table 2. *List of queries and their corresponding costs*

For each query, the negotiation processing computes the trade-off that minimizes the query cost and maximizes the quality over the four sources with eventually renegotiating quality contracts to find the best trade-off.

For this example, as illustrated in Figure 6, starting in state *i*, the new state *k1* (provided by source S4) is accepted, but the new state *k2* (provided by source S1) is only accepted with a certain probability. The probability of accepting a worse state is high at the beginning and decreases at the temperature decreases. Finally the global minimum of trade-off is found and is provided by source S1 that will be queried first.

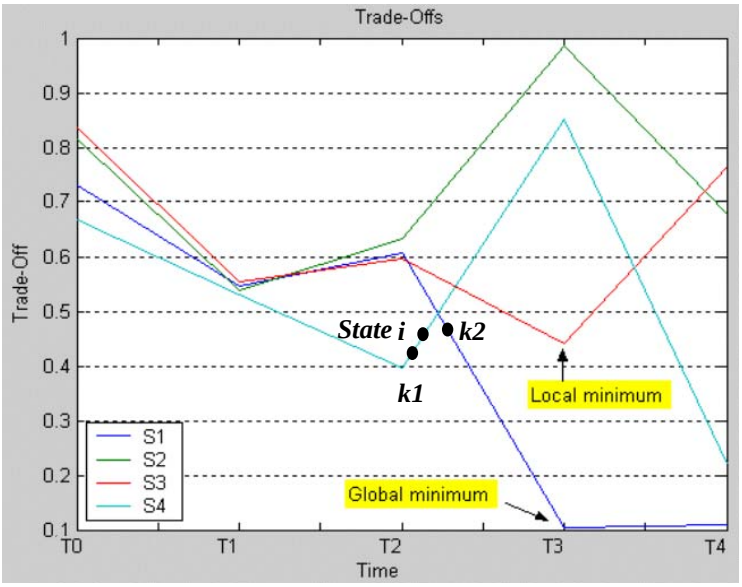

Figure 6. *Reaching the global minimum trade-off for the quality-extended query q1*

6. CONCLUSION

We propose XQuaL, a query language extension including declarations of quality contracts and profiles and we present the algorithm of quality-driven negotiation for adaptive query processing. Our negotiator module is fully implemented and preliminary performance experiments are ongoing.

The contribution of this work is twofold. Firstly, we propose a declarative language for specifying and querying data as well as quality metadata: we take advantage of QML [9] in the domain of Quality of Service for flexibly extending our query language with constraints on quality of service and on quality of distributed data sources. Secondly, past work on query processing has largely ignored the issue of source quality negotiation and trade-offs between query costs and result quality gains: our approach introduces trade-offs into the query processing considering the multi-dimensional quality aspect.

Our current work is focused on the definition of appropriate database statistics and a cost model, and we describe plan enumeration including heuristics and different strategies for quality negotiation. In most of the real world, it is quite natural that quality–extended query should meet a number of different and conflicting quality dimensions. Optimizing a particular objective function may sacrifice optimization of another dependent and conflicting objective. The purpose of our future work is to study the quality-adaptive query processing problem from the perspective of multi-objective optimization.

7. BIBLIOGRAPHY

[1] Bochmann G., and Hafid A., Some principles for quality of service management, *Distributed Systems Engineering Journal*, vol. 4, 1997, pp.16-27.

[2] Ballou D.P., Pazer H., Modeling completeness versus consistency tradeoffs in information decision contexts. *IEEE TKDE*, 15(1): 240-243, 2002.

[3] Carlson D., Data Stewardship in action, *DM Review*, May 2002.

[4] Clavanese D., DeGiacomo G., Lenzerini M., Nardi D. and Rosati R., Data integration in Datawarehousing, Tech. Rep. DWQ-UNIROMA-001, DWQ Consortium, 1997.

[5] Cui Y., Widom J. Lineage Tracing for general data warehouse transformation. *Intl. Conf. on Very Large Databases*, 2001, p. 471-480.

[6] Dasu T., Johnson T., *Exploratory Data Mining and Data cleaning*, Wiley, 2003.

[7] Dasu T., Johnson T., Muthukrishnan S., Shkapenyuk V., Mining database structure or, how to build a data quality browser. *Intl. Conf. ACM SIGMOD*, 2002.

[8] Fox C., Levitin A., Redman T., The notion of data and its quality dimensions, *Information Processing and Management*, vol. 30, no. 1, 1994.

[9] Frølund S. and Koistinen J., QML: A Language for Quality of Service Specification. Tech. Rep. HPL-98-10, Software Technology Laboratory, Hewlett-Packard, 1998.

[10] Frølund S. and Koistinen J., Quality of Service Aware Distributed Object Systems, *5th USENIX Conf. on Object-Oriented Technologies and Systems*, 1999.

[11] Fayyad U., Piatetsky-Shapiro G., Smyth P., Uthurusamy R. (Ed.), *Advances in Knowledge Discovery and Data Mining*, AAAI Press, MIT Press, 1996.

[12] Galhardas H., Florescu D., Shasha D., and Simon E., An Extensible Framework for Data Cleaning, *Intl. Conf. on Data Engineering*, 2000.

[13] Goodchild M., Jeansoulin R. (Ed.), *Data quality in geographic information: from error to uncertainty*, Hermès, 1998.

[14] Hernandez M., Stolfo S., Real-world data is dirty: data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9-37, 1998.

[15] Hou W.C., Zhang Z., Enhancing database correctness: a statistical approach. *Intl. Conf. ACM SIGMOD*, 1995.

[16] Kahn B.K., Strong D.M., Wang R. Y., Information quality benchmarks: product and service performance. *Com of the ACM*, 45(4): 184-192, 2002.

[17] Kirkpatrick, S., Gerlatt, C. D. Jr., and Vecchi, M.P., Optimization by Simulated Annealing, Science 220, 671-680, 1983.

[18] Lee M. L., Lu H., Ling T.W. and Ko Y.K., Cleansing Data for Mining and Warehousing, *Intl. Conf. on Database and Expert Systems, Applications*, 1999.

[19] Maletic J., Marcus A., Data Cleansing: Beyond Integrity Analysis, *Intl. Conf. on Information Quality* (ICIQ2000), Massachusetts Institute of Technology, Boston, 2000, p. 200-209.

[20] Mihaila G. A., Raschid L., and Vidal M., Using quality of data metadata for source selection and ranking. *WebDB Workshop*, 2000, p. 93-98.
[21] Naumann F., *Quality-Driven Query Answering for Integrated Information Systems*, LNCS 2261, Springer-Verlag, 2002.
[22] Naumann F., Leser U., and Freytag J., Quality-driven integration of heterogeneous information systems. *Intl. Conf. VLDB*, 1999.
[23] Parsaye K., Chignell M., *Intelligent Database Tools and Applications*, John Wiley, 1993.
[24] Paradice D.B., Fuerst W.L., An MIS data quality management strategy based on an optimal methodology. *Journal of Information Systems*, 5(1):48 - 66, 1991.
[25] Pipino P., Lee Y.W., Data quality assessment. *Com. of the ACM*, 45(4): 211-218, 2002.
[26] Raman V., Hellerstein J. M., Potter's wheel: an interactive data cleaning system, *Intl. Conf. VLDB*, 2001.
[27] Redman T., *Data quality for the information age*, Artech House Publishers, 1996.
[28] Rothenberg J., Metadata to support data quality and longevity, *IEEE Metadata Conf.*, 1996.
[29] Scannapieco M., Naumann F. (Eds), *Intl. ACM SIGMOD Workshop on Information Quality in Information Systems*, 2004.
[30] Schlimmer J., Learning determinations and checking databases, *AAAI-91 Workshop on Knowledge Discovery in Databases*, 1991.
[31] Schlimmer J., Self-modeling databases, *IEEE Expert*, 8(2):35-43, 1993.
[32] Strong D., Lee Y., Wang R., Data quality in context, Com. of the ACM, 40(5), p. 103-110, 1997.
[33] Sheth A., Wood C., Kashyap V., Q-data: Using deductive database technology to improve data quality, Kluwer Academic Press, 1993, p. 23-56.
[34] Vassiliadis P., Data Warehouse Modeling and Quality Issues, PhD thesis, Department of Electrical and Computer Engineering, National Technical University of Athens (Greece), 2000.
[35] Vassiliadis P., Vagena Z., Skiadopoulos S., Karayannidis N., ARKTOS: A Tool For Data Cleaning and Transformation in Data Warehouse Environments. *IEEE Data Eng. Bull*,. 23(4): 42-47, 2000.
[36] Wang R., A product perspective on Total Data Quality Management, *Com. of the ACM*, 41(2): 58-65, 1998.
[37] Wang R., Journey to Data Quality, vol. 23 of *Advances in Database Systems*, Kluwer Academic Press, Boston, 2002.
[38] Winkler W., Data Cleaning Methods, *Intl. Conf. KDD*, 2003.

8. APPENDICES: SYNTAX OF XQUAL

Qwith-query ::=	query QWITH declarations ;
query ::=	(query) | query set-op query | query bool-op query | NOT query | query constraintop query | query IN query | COUNT (query) | constant | var | sfw-query | EXISTS var query : query | FORALL var query : query
set-op ::=	UNION | INTERSECT | MINUS
bool-op ::=	AND | OR
constant ::=	integer literal | real literal | quoted string literal | true | false
var ::=	IDENTIFIER | $IDENTIFIER | @IDENTIFIER
sfw-query ::=	sfw-query WHERE query | SELECT projs-list FROM ranges-list
projs-list ::=	projs-list, var | * | var
ranges-list ::=	ranges-list, one-range | one-range
one-range ::=	var query | path-expr
path-expression ::=	path-elem.path-expression | path-elem
path-elem ::=	[from-to]query | query | path-elem[from-to]
from-to ::=	from-to:query | var
declarations ::=	declarations declaration ;
declaration ::=	contractTypeDeclaration | contractDeclaration | profileDeclaration
contractTypeDeclaration ::=	TYPE IDENTIFIER = contractType
contractType ::=	CONTRACT { dimensions } ;
dimensions ::=	dimensions dimension | dimension
dimension ::=	dimName : dimType ;
dimName ::=	IDENTIFIER
dimType ::=	dimSort | dimSort unit

items ::=	items , IDENTIFIER | IDENTIFIER
dimSort ::=	ENUM { items } | relSem ENUM { items } WITH order | SET { items } | relSem SET { items } | relSem SET { items } WITH order | relSem NUMERIC
relations ::=	relations , relation | relation
relation ::=	IDENTIFIER < IDENTIFIER
order ::=	ORDER { relations }
unit ::=	unit / base-unit | base-unit
base-unit ::=	% | IDENTIFIER
relSem ::=	DECREASING | INCREASING
contractDeclaration ::=	IDENTIFIER = contractExpression
contractExpression ::=	IDENTIFIER contractDefinition | IDENTIFIER REFINED BY {constraints }
contractDefinition ::=	CONTRACT { constraints }
constraints ::=	constraints constraint | constraint ;
constraint ::=	dimName constraintOp dimValue | dimName { aspects }
constraintOp ::=	== | >= | <= | > | < | LIKE | !=
dimValue ::=	literal unit | literal
literal ::=	IDENTIFIER | { items } | NUMBER
aspects ::=	aspects aspect;
aspect ::=	PERCENTILE NUMBER constraintOp dimValue | MEAN constraintOp dimValue | VARIANCE constraintOp dimValue | FREQUENCY freqRange constraintOp NUMBER %
freqRange ::=	dimValue | lRangeLimit dimValue , dimValue rRangeLimit
lRangeLimit ::=	[| (
rRangeLimit ::=	] |)
profileDeclaration ::=	IDENTIFIER FOR IDENTIFIER = profileExpression
profileExpression ::=	profile | IDENTIFIER REFINED BY { requisites }
profile ::=	PROFILE { requisites }
requisites ::=	requisites requisite ;
requisite ::=	REQUIRE contractList | FROM entityList REQUIRE contractList
contractList ::=	contractList , contractElem | contractElem
contractElem ::=	IDENTIFIER | contractExpression
entityList ::=	entityList , entity | entity
entity ::=	IDENTIFIER | IDENTIFIER . IDENTIFIER | RESULT OF IDENTIFIER