Quality of Service and Optimization in Data Integration Systems

Reinhard Braumandl*
University of Passau

Abstract: Due to the ever increasing impacts of globalization, people will/have to work on data which is distributed all around the world [LKK+97]. Query processing on the corresponding data sources is a key data processing task in most distributed applications and rather difficult to implement for interactive usage in an Internet scale environment. ObjectGlobe is a data integration system which enables interactive query processing in such an environment. The basic architecture of the ObjectGlobe system and its integrated quality of service (QoS) management component were developed in this dissertation and briefly summarized in this paper. The dissertation [Bra02] itself can be downloaded at
`http://elib.ub.uni-passau.de/opus/volltexte/2002/27`.

Advisors: Prof. Alfons Kemper, Ph.D. (University of Passau)
Prof. Dr. Donald Kossmann (Technical University of Munich)

1 Introduction

The proceeding globalization we encounter today results in a demand for global data processing. The Internet as the largest global computer network is the most prominent enabler for global data processing. Query processing is one task in this field and it plays an important role for many application domains, e.g., applications which depend on some kind of decision making. In this respect, the Internet provides access to a large number of interesting data sources like hotel listings, flight schedules, stock rates, earth observation images or genome databases. Data integration or mediator systems were developed to access these distributed and heterogeneous data sources on the Internet. First, these systems were closed systems with a predetermined, fixed set of data sources and a centralized architecture with respect to the execution of mediator related tasks. More recently, flexible, distributed data integration systems were conceived, e.g., Amos II [JKR99] or our ObjectGlobe system [BKK+01]. In the following paragraphs, we describe the ObjectGlobe architecture which represents an open and distributed data integration system.

Within an ObjectGlobe federation three kinds of providers that can participate in a corresponding information economy are distinguished: *data providers* which supply data, *func-*

*current affiliation: Allianz Versicherungs AG

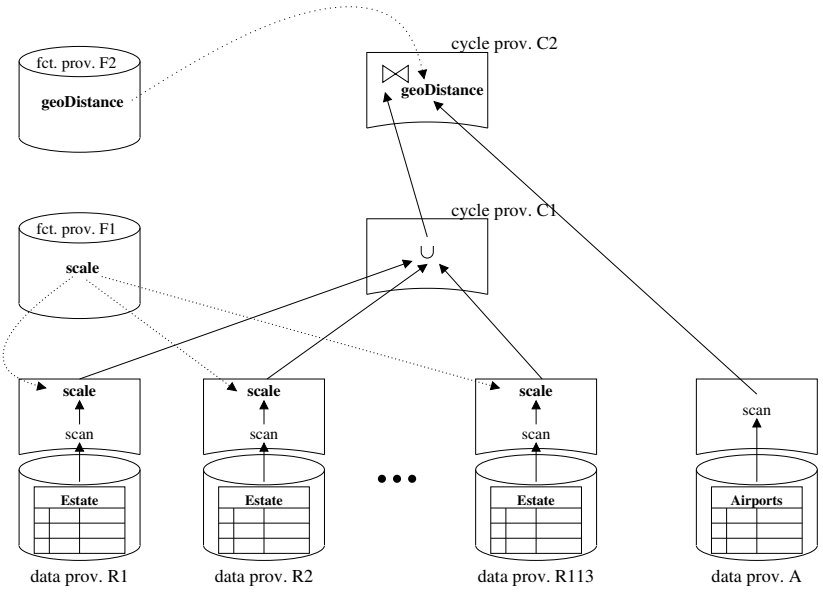

Figure 1: An Example Query Execution in an ObjectGlobe Federation.

tion providers which offer operators and functions to process data, and *cycle providers* which are contracted to execute operators and functions. In ObjectGlobe, the services of these kinds of providers can be combined in an almost orthogonal way; exceptions just arise for security, privacy and capacity reasons. Therefore, ObjectGlobe enables an open and distributed query processing services market.

For example, suppose that an ObjectGlobe user is searching for a villa in the Mediterranean area with requirements regarding the maximum distance to the next airport and the amount of building area. The notional SQL-query for this application computes a join between real estate information and data about airports. The join predicate uses a user-defined, external function which computes the length of the bee-line between two locations. The query also uses an external operator (named **scale**) for scaling the images of the estate offers into a handy size. The meaning of the attributes of our real estate relation should be obvious.

```
select e.Price, e.Location.City,
       scale(e.Image,0.3), a.Name
from Estate e, Airports a
where e.building-area > 200 and
  geoDistance(e.Location,a.Location) < 10
  and e.Location.region = 'Mediterranean';
```

In an ObjectGlobe federation, external data sets are incorporated by so-called wrappers and internally represented in a nested relational format. Hence, the data of each European realtor could be incorporated as a partition of such a relation, here named *Estate*. An example query execution in an ObjectGlobe federation is shown in Figure 1. The data

for airports is contributed by a data provider **A** and for real estate by the data providers **R1, . . . , R113**, respectively. The providers for real estate information represent combined data/cycle providers, executing the external function **scale** which is provided by the function provider **F1**. Additionally, two pure cycle providers execute the **union** operation on the real estate partitions and the **join** operation between real estate and airports information. Pure cycle providers play an important role when data providers cannot or are not willing to execute any query operators except the scan operators or when operations can be placed at cycle providers in a way which reduces communication costs.

Obviously, the effects of query executions in such an environment can hardly be overseen by a user. Therefore, a quality of service management component of ObjectGlobe supports user defined quality constraints [BKK03]. [Wei99] gives a good motivation for the need to integrate the handling of service quality guarantees in information systems. One aspect of QoS in a data integration system is certainly data quality. This topic has already been tackled in the literature, for example, in [NLF99].

2 The Quality of Service Model

As we have seen in the introduction, in an information economy a user should be able to constrain the relationship between the qualities of the result and the execution itself and the costs for the services of providers. For example, users would then be able to express that they are willing to pay more for the execution of a query, if the query finishes earlier.

This means, that analogously to cost models in traditional database systems, we need a model for our quality constraints in order to describe and assess the quality of queries, query evaluation plans and query executions.

Therefore, we need a set of QoS parameters which can be used for declaratively specifying QoS constraints for a query execution. In the query processing context QoS parameters can belong to three different dimensions: the result quality of the query, the duration/timeliness of the query execution and its monetary cost.

Parameters for the Query Result Quality:

- the oldest time stamp of the last update for a partition p of of a used relation S or its maximum staleness factor.
- the share of the used partitions in respect to the whole data of a relation S. This parameter is also called completeness.
- a lower bound for the result cardinality. This parameter can be used to express that the user expects a minimum number of result tuples.
- an upper bound for the result cardinality. Such a parameter corresponds to a *stop after* clause, whose support in query optimization and execution has already been studied in the literature [CK98].

Parameters for the Query Execution Time: The execution of the query is characterized by the time spent in different phases of the execution of a plan:

- the time spent in the `open`-phase of a plan. In an iterator based [Gra93] query engine like ours, this is roughly the time from the start of the query execution until the first tuple can be delivered.
- the time for producing all the result tuples of the plan starting at the point, when the open-phase has finished and ending, when the last tuple has been produced by the query. This phase is called the `next`-phase.

Parameters for the Query Evaluation Cost: Since providers can charge for their services in our information economy, a user should be able to specify an upper bound for the respective consumption by a query. Therefore, the quality parameters regarding the cost of a query take into account:

- the cost for services of function providers, i.e., the cost for leasing a function for the duration of the query.
- the cost for services of data providers, i.e., the cost for reading the data at the respective data providers.
- the cost for services of cycle providers, i.e., the cost for executing parts of the query at foreign cycle providers.

3 The Integration of QoS Management in Query Processing

Naturally, the ability to guarantee QoS constraints depends on the service quality guarantees one can get from the underlying shared resources. Among the common scheduling disciplines *best effort*, *priority-based scheduling*, and *reservation*, only the latter allows to construct a QoS management which can absolutely guarantee the QoS constraints for an accepted query. But reservation normally entails over-booking and inefficient resource utilization and is therefore rarely used for scheduling computer or network devices. Due to this fact, there remain two obvious goals for QoS management in our context:

- The percentage of queries, whose quality constraints could be fulfilled, should be maximized. This percentage is calculated based on the overall number of queries which are issued and not only on the number of those for which a constraint compliant query plan could be determined.
- The execution of queries which cannot fulfill their QoS constraints, should be stopped as early as possible. In this way the query does not waste the time an money of the user anymore. Of course, if the missed quality constraint is a soft one, the query should not be stopped but executed in a best-effort manner.

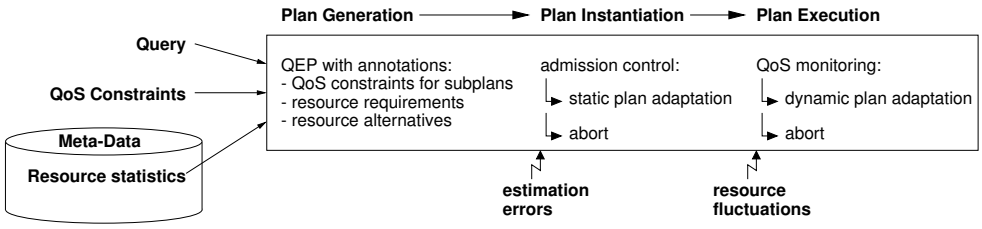

Figure 2: The Interaction of Query Processing and QoS Management

An overview of query processing with integrated QoS management is depicted in Figure 2. The starting point for query processing in our system is given by the description of the query itself, the QoS constraints for it and statistics about the resources which can be used for processing the query. In our scenario cycle providers, the partitions from data providers and the functions of function providers belong to these resources together with all the network links connecting them.

Figure 2 also shows the activities of our QoS management during the query processing phases of plan generation, instantiation and execution.

Plan Generation: The optimizer generates a query evaluation plan (QEP) which contains information about the used data, cycle and function providers and about the way their services are combined to compute a specific query. The optimizer itself is in essential an intelligent search routine which is in many cases and also in ours dynamic programming based. It assesses a huge number of different plans with different providers by a quality model which provides formulas for estimating the quality parameters based on the structure of the plan and statistics about the providers. Hereby, every considered plan is constructed piecewise in a bottom-up manner and for every sub-plan which appears in such a process its quality parameters are computed with the quality model. Only a plan which fulfills all the user constraints is considered for the later phases and its plan description will be annotated with the quality estimations and resource requirements for every sub-plan. Additionally, if the optimizer can find approximately equivalent alternatives for resources used in the query evaluation plan, these are also annotated in the plan.

Plan Instantiation: During plan instantiation, sub-plans are distributed to cycle providers, functions are loaded from function providers and connections to data providers are established. When a sub-plan of a query uses the service of a specific provider, it is checked, if the resource requirements resulting from the quality constraints for that sub-plan can be satisfied by this provider. For a cycle provider this would mean that if the optimizer underestimated the load on the respective cycle provider the newly arrived sub-plan will probably not be able to meet its constraints. Furthermore, all the other sub-plans on that cycle provider would be in danger of missing their quality constraints, if we execute the new sub-plan there. As a result of this admission control, the execution of the new sub-plan would be rejected or, if possible, the

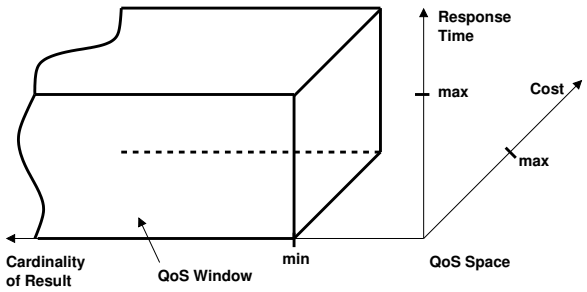

Figure 3: The QoS Space and the QoS Window.

sub-plan will be adapted.

Plan Execution: During query execution, fluctuations regarding resource availability for, e.g., CPU time or network bandwidth and again estimation errors by the optimizer might violate the constraints on the quality parameters. In order to detect these violations, a monitoring component traces the current status of these parameters for every relevant sub-plan of the query. If this component detects a potential violation of the quality constraints for a sub-plan, it first tries to adapt the sub-plan so that it will meet its constraints again, or if this is not possible, it will abort the execution of this sub-plan. The plan adaptations during the instantiation phase can be performed rather easily because the plan is not instantiated yet. Here we need adaptations which can be applied also after a sub-plan has started to execute.

By estimating the necessary quality constraints for sub-plans of a QoS compliant query plan the QoS management can monitor the development of the quality parameters on a fine granularity. This helps in detecting potential quality misses quite early. For example, if there are pipeline-breaking operators in the plan, most of the work for the query could have already been done, when the first tuples arrive at the top of the plan. In our case, the plan beneath the pipeline breaker has its own quality constraints which must be monitored and enforced by the QoS management.

4 Quality of Service Enhanced Plan Generation

In this section we mention the necessary modifications of a classic, dynamic programming based query optimizer for supporting QoS constraints during plan generation. We concentrate on the description of those parts of the optimization process which play an important role for QoS management and thus need modifications compared to their standard form.

Selecting Providers In many cases, it will not be feasible to consider all possible data providers for a given data set because the resulting amount of data processed in a widely

distributed environment would result in unacceptable running times. Thus, an additional task for QoS management is to select the most relevant data providers and find appropriate cycle providers which are able to efficiently process the data from the selected data providers. Since compact query evaluation plans will also be rather efficient in a large scale environment, we use clustering algorithms to group data and cycle providers with respect to their 'network distance' to each other. During optimization it is often sufficient to work on the resulting clusters instead of single providers. In this way, we can avoid the combinatorial explosion in finding appropriate subsets of providers.

Estimating QoS Parameters Query evaluation plans are constructed in a modular way using basic operators such as join, union, or user-defined operators. Analogously, the cost (and other properties) of plans are computed in a modular way using cost functions for the individual operators. For QoS management, an extended framework is needed in order to construct plans and estimate the properties of plans in such a modular way. In our framework, these computations are based on formulas for every operator which determine the amount of work the operator has to perform at the phases given by the iterator model. The information about the distribution of operators and parallelism properties of the plan construction are then used to compute the corresponding effects on the overall timing of a plan.

Pruning Query Evaluation Plans The query optimizer enumerates alternative plans and prunes inferior plans based on their properties (e.g., estimated cost). A QoS-enhanced optimizer requires a special pruning metric in order to take all QoS parameters of a query into account. This metric is needed because we inherently use multi-objective optimization in the QoS case. The quality dimensions span a space which we call QoS space, and the user-defined constraints determine an area in that space which we call QoS window. This is shown in Figure 3 for the (simplified) three dimensional QoS space. During optimization every enumerated plan is mapped on a point in that QoS space by estimating the value for every quality parameter which appears in the quality model. Only plans which lie within the QoS window, fulfill the constraints of the user.

5 Adaptation of a Query Execution Plan

Query evaluation plans in our system have a "natural" fragmentation which is given by the thread and machine boundaries which appear in the instantiated operator tree. At these boundaries, monitor operators trace the actual quality parameters of their input plans and forecasts these parameters for the end of execution. The corresponding optimizer estimates for the respective sub-plan (as shown in Figure 4) represent the target values for the forecasted values and a comparison of these values shows if the corresponding sub-plan is still within its quotas.

When a violation of the QoS constraints is expected during the runtime of a query, we can try to counteract this violation by adapting the query execution plan. The adapta-

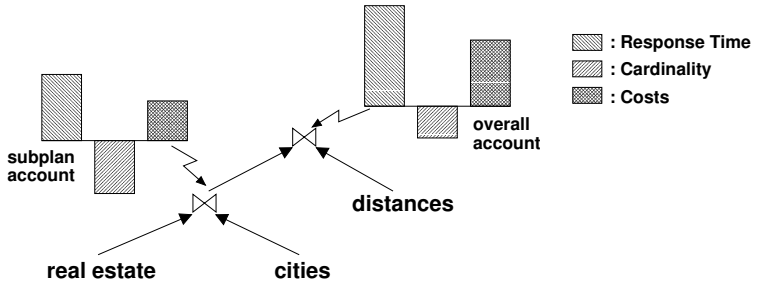

Figure 4: QoS Accounts of a Query Plan.

tions which we employ in our system to react to predicted QoS violations, operate on the resource allocation of the query evaluation plan. The corresponding resources are, for example, cycle providers, partitions from data providers and functions from function providers.

Example adaptations are:

increasePriority/decreasePriority adapt the priorities of threads which execute queries on ObjectGlobe servers. By this adaptations, we can speed up queries which need to finish sooner and we can slow down queries which currently seem to meet their time limits easily and can therefore free resources for queries with critical constraints.

useCompression activates on the fly compression for network links which seem to be slower than expected.

movePlan can be used to move a sub-plan of a query from one cycle provider to another cycle provider. Again, this adaptation can be performed on the fly in an ObjectGlobe federation, even if the plan has already begun to work.

6 Controlling Adaptations

Plan adaptation during runtime is controlled by monitor operators because they are placed at the most interesting positions for adaptations in the query evaluation plan and also gather most of the information which is needed for this task. As shown in Figure 5, a monitor operator uses a rule-based fuzzy controller which gets the forecasts of the quality parameters and other state descriptions of the respective plan fragment as input. With this information the fuzzy controller determines, if an adaptation should be applied and what adaptation this should be.

One reason for the use of a fuzzy controller in our system is that our runtime adaptations are discrete and this is quite easy to support in the inference rules which form the basis of the fuzzy controller's decisions. Furthermore, estimation errors and resource fluctuations introduce some uncertainty factors in the control process which can be modeled by

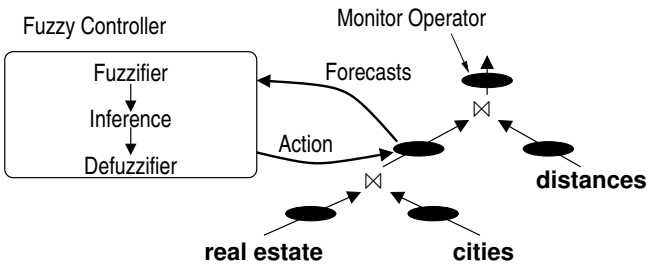

Figure 5: Feedback Loop for QoS Adaptations.

fuzzy logic [KY95] quite easily. The inference rules of a fuzzy controller also provide a highly configurable means to incorporate expert knowledge for the adaptation process in a very intuitive way. Therefore, it becomes possible to experiment with varying adaptation strategies by just changing the inference rules and perhaps the definition of the underlying fuzzy sets.

A fuzzifier transforms the numeric input values into *linguistic values* of *linguistic variables*. For each input variable of the controller there exists one such *linguistic variable*. This transformation is depicted for the forecasted cost parameter in Figure 6. The inference rules of a fuzzy controller works on linguistic variables and values. Such a rule is of the form

if X_1 is A_1 and $\ldots$ and X_n is A_n **then** Y is B

where $X_1, \ldots, X_n$ and Y are linguistic variables and $A_1, \ldots, A_n$ and B are linguistic values with accompanying fuzzy sets. A small portion of an example rule set is shown below. Here *lfc*, *lfr* and *lft* are the linguistic variables for cost-, result- and time quality parameters; *lep*, *lbp* and *lss* correspond to the execution progress, the buffer pressure and the state size.

if *lfc* is *endangered* and *lep* is *late* **then** *abort* is *preferred*

if *lft* is *endangered* and *lbp* is *high* and *lss* is *small* **then** *movePlan* is *preferred*

if *lft* is *endangered* and *lep* is *middle* and *lbp* is *high* **then** *increasePriority* is *possible*

if *lfr* is *endangered* and *lep* is *early* and *lfc* is *compliant* **then** *addSubPlan* is *preferred*

7 Acknowledgements

First of all, I have to thank my advisors Prof. Alfons Kemper and Prof. Donald Kossmann for their support. They gave me the opportunity to participate in an ambitious and visionary project. I have learned a lot from their insight and experience in doing research work. Their advices provided invaluable guidance for my work.

I also wish to express my gratitude to the whole ObjectGlobe team at the University of Passau for many helpful discussions and for the pleasant working atmosphere

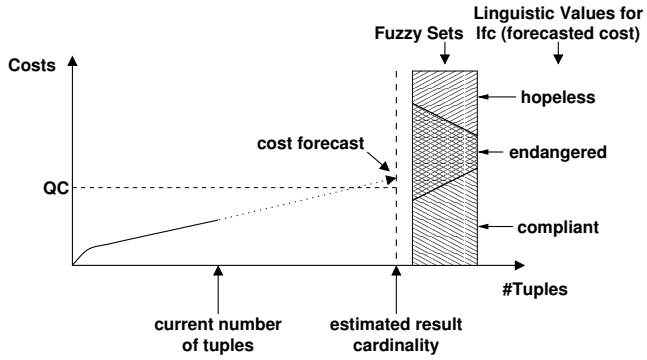

Figure 6: Mapping Quality Parameter Forecasts on Fuzzy Sets.

Literaturverzeichnis

[BKK+01] R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam, and K. Stocker. ObjectGlobe: Ubiquitous query processing on the Internet. *The VLDB Journal: Special Issue on E-Services*, 10(3):48–71, August 2001.

[BKK03] R. Braumandl, A. Kemper, and D. Kossmann. Quality of service in an information economy. 2003. Submitted for publication.

[Bra02] R. Braumandl. *Quality of Service and Optimization in Data Integration Systems*. PhD thesis, Universität Passau, Fakultät für Mathematik und Informatik, D-94030 Passau, 2002. Universität Passau.

[CK98] M. Carey and D. Kossmann. Reducing the braking distance of an SQL query engine. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 158–169, New York, USA, August 1998.

[Gra93] G. Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2):73–170, June 1993.

[JKR99] V. Josifovski, T. Katchaounov, and T. Risch. Optimizing queries in distributed and composable mediators. In *Proc. of the IFCIS International Conference on Cooperative Information Systems*, pages 291 – 302, Edinburgh, Scotland, 1999.

[KY95] G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic*. Prentice Hall, 1995.

[LKK+97] P. Lockemann, U. Kölsch, A. Koschel, R. Kramer, R. Nikolai, M. Wallrath, and H.-D. Walter. The network as a global database: Challenges of interoperability, proactivity, interactiveness, legacy. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 567–574, Athens, Greece, August 1997.

[NLF99] Felix Naumann, Ulf Leser, and Johann Christoph Freytag. Quality-driven integration of heterogenous information systems. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 447–458, Edinburgh, GB, September 1999.

[Wei99] G. Weikum. Towards guaranteed quality and dependability of information services. In *Proc. GI Conf. on Database Systems for Office, Engineering, and Scientific Applications (BTW)*, Informatik aktuell, New York, Berlin, etc., 1999. Springer-Verlag.