Quality Support in Database Systems

Yi-Cheng Tu
Department of Computer Sciences, Purdue University
West Lafayette, IN, 47907, USA

Abstract

This paper addresses the quality features of (continuous) queries on various data management systems. System support of quality of queries has attracted little attention. We present some of the research problems related to this issue as well as tentative solutions to these problems, including a heuristic algorithm for optimal assignment of quality parameters to multiple queries. We also discuss the design and implementation of quality-aware query processing frameworks in various DBMSs.

Keywords: Boston, quality, QoS, resource allocation, data streams

1 Introduction

Emerging data management applications such as data streams, multimedia, and sensor networks have brought up many new challenges to the database community. Unlike traditional data processing, queries to these systems are generally of a continuous manner and allow different levels of timeliness, reliability, and fuzziness. These temporal/spatial constraints are generally referred to as *Quality* features of the query, which are analogous to *Quality of Service* (QoS) parameters in multimedia applications. From the point of view of DBMS design, not only do we have concerns on the *correctness* of the data, query processing according to appropriate *quality* requirements is important as well.

An (incomplete) list of quality parameters for various DBMSs is shown in Table 1. Although this paper focuses on streaming/sensor databases, discussions on multimedia and relational databases are helpful in our understanding of the problem. First of all, quality features of multimedia data[1] (QoS) have long been defined and their OS-level support extensively studied [1, 2]. However, QoS support has to be taken into account from the application (e.g. multimedia database) level as human perception of media content is the primary concern in such systems [3]. In a quality-aware multimedia DBMS, query processing is accomplished in two stages: direct or content-based search and delivery of search results [4]. Quality issues are only brought up at the second stage. Quality parameters in streaming/sensor databases are defined closely related to the spatial/temporal resolutions found in those of multimedia databases. The difference is that quality features have to be considered at all stages of query processing for the streaming/sensor DBMSs. Quality parameters in such DBMSs can be divided into two categories: 1. *Quality of Data* (QoD) that captures how frequently and accurately the data are collected. Parameters in this category include data freshness (updating frequency) and loss (shedding) rate; 2. *Quality of Query* (QoQ): requirements specified by the user on the query processing such as query frequency, delay, and precision. For the case of a relational database dealing with tradi-

[1] To be specific, video data.

Table 1: Quality Parameters in Database Systems.

System	Quality Parameters
Multimedia Database	*Frame Width, Frame Height, Color Resolution, Time Guarantee, Signal-to-noise ratio, Security*
Streaming/Sensor Database	*Frequency, Data Freshness, Precision, Delay*
Relational Database	*Processing Time*

tional data, the query optimization is generally towards minimizing *processing time*, which can also be regarded as a quality parameter (in a broader sense).

There are two major problems in quality support of DBMSs:

1. Most DBMSs are supposed to handle multiple queries concurrently, especially continuous queries in sensor/streaming databases. Running all queries at their best quality is not an option due to resource constraints. The assignment of quality to individual queries is essential in gaining better system-level performance.
2. Once the quality parameters of a query are set, how do we guarantee the query is processed following these quality features?

Solutions to problem 2 have been carried out by previous research on both multimedia [1] and data management systems [5]. The Ph.D. work concentrates on problem 1.

Quality support in databases is basically a problem about optimizing resource allocation among multiple queries. System resources of interest include CPU cycles, memory, storage, and network bandwidth, depending on the application studied. One rule for the choice of resource is that we are only interested in those that could become a bottleneck. Generally, higher quality means more system resources consumed. As no system has unlimited resource, an important goal of our research is to develop smart resource distribution algorithms that achieves highest possible system performance (benefit). Thus, quality support has to be studied at both application and system(resource) levels. The interaction between them, namely, the mapping of application quality parameters to resources, is also indispensable [6].

2 Our Approaches

With the assumption of the availability of resource scheduling modules (problem 2 above), we propose the design of a *Quality Manager* (QM) as an augment of the current query optimization module in the DBMS. Details of one such QM can be found in [6]. The QM is responsible for admission control and making static/dynamic decisions on the quality of admitted queries. The resources are allocated based on these decisions (via quality-to-resource mapping) by the underlying scheduling mechanisms.

2.1 Quality Assignment Algorithm

Problem 1 in Section 1 can be formulated as the following: Consider a system with a total number of m quality dimensions and n resources. We denote a set of quality parameters as $\vec{q} = (q_1, q_2, \ldots, q_m)$ with q_i being the quality on the i-th dimension and a set of resource consumption values as $\vec{r} = (r_1, r_2, \ldots, r_n)$. The total amount of the j-th resource available from the system is R_j. The user satisfaction, or benefit to system, associated with a quality vector $\vec{q}$ is captured by a *utility function* $U : \vec{q} \rightarrow \Re$. We have $U(\vec{q}) = \sum_{i=1}^{m} w_i u_i(q_i)$ where w_i is a weight for

i-th quality ($\sum_{i=1}^{m} w_i = 1$) and u_i the single-dimension utility function. We assume u_i is a non-decreasing, convex continuous function of q_i. For each type of queries, we have a set of *Quality Mapping Functions* $f_j : \vec{q} \rightarrow \Re$ ($1 \leq j \leq n$) to calculate the resources required by a certain quality vector. In a system with N queries, the problem is to get the $\vec{q}_k$ for all k ($k \in [1, N]$) queries that maximizes total utility $\sum_{k=1}^{N} U_k(\vec{q}_k)$ subject to $\sum_{k=1}^{N} f_j(\vec{q}_k) \leq R_j$ for all $j \in [1, n]$ (resource constraints). For each query, all values of u_i, w_i, and all mapping functions f_j are known, as part of the *query profile*.

The assignment of discrete quality values to concurrent queries is a NP-hard optimization problem [7]. For our problem with continuous quality parameters, the complexity of obtaining a numerical solution by direct search is $O\big(Nm \log C\big)$ where C is a constant (proof not shown). This is obviously unacceptable since N could be large and the optimization has to be run very frequently at runtime. We propose a fast heuristic algorithm that gives solutions reasonably close to optimum. The focal point of this algorithm is an economic quality evaluation model. For each type of queries, we define the *Cost Efficiency* of being executed under a quality vector $\vec{q}$ as:

$$CE(\vec{q}) = \frac{U(\vec{q})}{C(\vec{r})} \quad (1)$$

where $\vec{r}$ is the resource vector derived from the the Quality Mapping Functions and C the *cost function* of $\vec{r}$. In the following paragraphs, we discuss the exact quality assignment problems and how Eq (1) can be used to optimize performance in various systems.

2.2 Streaming/Sensor Data Management Systems

In a streaming/sensor data management system (Figure 1), there are two types of computing activities that consume resources continuously: continuous queries and data streams, both of which have quality descriptions: QoQ and QoD (Section 1). The interesting feature of QoD is that they are not configurable by modifying resource allocation to the queries even though they are part of the quality parameters. The reason is that a query may access data from any streams.

We view the quality control in such systems as a hierarchical decision-making problem: 1. Periodically determine the amount of resource used for data streams vs. the amount for (continuous) queries; 2. Assignment of QoQs to all queries, this has to be reevaluated when queries enter/leave the system or decision 1 leads to a resource reallocation; 3. Assignment of QoDs to all data streams.

Decision 1 can be made based on the following: we record the dimensional utility values (u) of all quality parameters for all (or part) of the queries. The mean of *quality skewness*, defined as $\frac{u(QoD)}{u(QoQ)}$, is used as a sensor to adjust the resource allocation. A high (low) skewness means more (less) resources need to be allocated to the queries. The challenge is to find an empirical formula that relates skewness to resource allocation. Similar solution to problem 3: those will a lower quality skewness and contributed more utility to the system deserve better QoD.

Decision problem 2 strictly follows the one described in Section 2.1. Our solution is a greedy algorithm based on a modification to Eq (1). We define the *Dimensional CE* (DCE) as $CE_j = \frac{U(\vec{q})}{r_j}$ where r_j is the j-th element in $\vec{r}$. The first order derivatives of DCEs ($\frac{\partial U}{\partial r_j}$) of all running queries are put into n priority queues, one for each resource. These values actually represent the instantaneous utility gain by consuming a specific resource. When a new query enters the system, it starts by taking resources from the queries at the tail of the queues. In addition, each

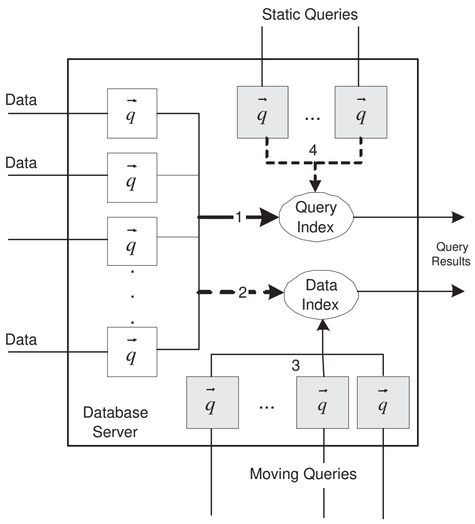

Figure 1: System model of a streaming/sensor data management system

step of resource exchange should lead to an increase of utility. If the new query cannot find one to take resource from, it is rejected. When a query leaves system, other queries at the head of the lists takes the resources it releases. The time complexity of this algorithm is $O\big(\log N \log C\big)$.

2.3 Multimedia Databases

In processing quality-aware queries to multimedia databases, we allow users to pick up the quality values for media delivery. The assumption here is that users always stick to his/her favorite set of quality parameters. Once the media is being delivered with a certain quality, they are generally not changed afterwards as people are reluctant to view media with ever-changing qualities. Media objects are replicated at various media servers and various tools are available to generate media with different quality offline and on-the-fly. Therefore, there are always many ways (execution plans) to service a query, even when the choice of quality allows little flexibility. The quality control problem now becomes the development of a *cost model* for evaluation of alternative execution plans for each query such that the system performance is maximized. System throughput (i.e. the number of concurrent queries) is the most frequently-used performance metric.

The QM of multimedia DBMS contains a *plan generator* that enumerates all possible plans for each query received. Our cost model evaluates query plans by applying Eq (1): the one with the highest CE value is chosen for execution. Note $U(\vec{q}) = 1$ for all plans but the Quality Mapping Functions (thus the $\vec{r}$ resulted) could be different for different plans. To calculate the cost of a resource vector, we propose two functions:

$$C(\vec{r}) = \max_{j=1}^{n} \left\{ \frac{H_j + r_j}{R_j} \right\} \tag{2}$$

and

$$C(\vec{r}) = \sum_{j=1}^{n} 2^{\frac{H_j}{R_j}} r_j. \tag{3}$$

Both equations take current usage of resource (H_j) into account. The first formula selects plans by minimizing the maximal resource use while the second one assigns a *price* to each resource based on its supply and demand.

2.4 Relational Databases

The case for handling traditional data is relative simple as both quality and resource vectors are of one single dimension. Generally, we only have concerns on the processing time and memory buffer is always the bottlenecking resource. The optimization problem formulated in Section 2.1 can be reduced to the following:

As $\vec{q}$ and $\vec{r}$ are both scalars, we denote them as q and r, respectively. Assuming f is an one-to-one mapping from q to r, then the utility can be defined as a function of r : $\mathcal{U}(r) = U(q) = U(f^{-1}(r))$. The problem becomes to find the memory allocation to each query (r_k) that maximizes $\sum_{k=1}^{N} \mathcal{U}_k(r_k)$ subject to $\sum_{k=1}^{N} r_k \leq R$. This is a problem with a known solution as the following:

$$\frac{\partial \mathcal{U}_1}{\partial r_1} = \frac{\partial \mathcal{U}_2}{\partial r_2} = \ldots = \frac{\partial \mathcal{U}_N}{\partial r_N}.$$

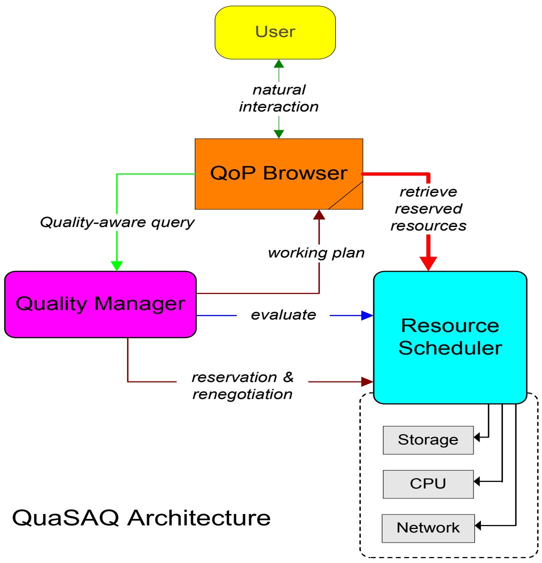

Figure 2: Architecture of the Quality-aware Query Processor in multimedia DBMS

The above status is also called the *Pareto equilibrium*. Theoretically, one can obtain a numerical value of the optimal r_k values in constant time.

3 Preliminary Results and Related Work

In our previous work [6], we propose the design and implementation of a QoS-Aware Query Processor (QuaSAQ) in multimedia databases (Figure 2) with an emphasis on the cost estimation of execution plans of QoS-enhanced queries. We assume a reservation-based resource management through which the resource allocation to a query is (highly) guaranteed throughout its lifetime. Experiments run on the QuaSAQ prototype show significantly improved QoS. System throughput using Eq (2) is 27% to 89% higher than that of a system using a randomized selection model. To the best of our knowledge, this is the first multimedia DBMS prototype that supports end-to-end QoS. The work on streaming/sensor DBMS is underway.

There are very few systematic studies on quality support and dynamic resource allocation in the context of streaming/sensor data management systems. Most relevant work can be found in the QoS control study of a streaming data management systems named Aurora [8]. Aurora follows a "dynamic data-static query" system model: query execution is accomplished by pushing data through a series of operators. Operators are of few types and can be shared by multiple queries thus a operator network is formed in a multi-query environment. In our model, both data and query are dynamic (we consider static queries as well). As a result, Aurora does not consider the control of QoQ parameters performed in our work.

The quality control (assignment) strategy in Aurora also has different focus from ours. When encountering degraded QoS, Aurora adjusts contention by dropping tuples (load shedding) [9]. This is more like an *adaptation* approach while our solution concentrates on regulating the resource to individual queries. Reference [5] discusses scheduling algorithms for CPU distribution into vertices in the Aurora operator network. However, it is not clear how the decisions on resource distribution are made according to QoS requirements.

One improvement of our work over Aurora is the high complexity of the quality control scenarios we deal with. In Aurora, the application QoS parameters are considered individually and the system resource studied only includes CPU cycles. Due to the lack of a combined view on multiple QoS dimensions and resources, their conclusions cannot be readily applied to the multi-QoS multi-resource scenario we consider in our research.

Similar in methodology to our research is the work by Davison and Graefe [10] who dynamically controls memory allocation to various join operators in relational databases. Again, they worked on an problem with one single quality parameter (processing time) and one resource (memory). Although they included I/O bandwidth in the research, its effects on system performance are always masked by those of memory.

Yu and Cornell [11] pointed out that traditional query/optimization strategies in relational databases could easily lead to poor system performance. This is due to the failure to capture the effects of resource allocation on performance. All efforts are dedicated to selecting a faster join plan and access path. They propose the concept of Return on Consumption (ROC) to guide the allocation of memory to queries:

$$ROC(m) = \frac{T(m_0) - T(m)}{mT(m) - m_0T(m_0)}$$

where m is memory use, m_0 the minimum memory allocation needed for the join, and T a mapping from memory to response time. This is similar in spirit to our cost-benefit model. The memory allocation strategy proposed in [10] is based on ROC.

4 Conclusions

Continuous queries on non-traditional data are very likely to be quality-critical. We believe that quality support under such scenarios should be an integrated functionality of the DBMS. In this paper we identified the challenges of quality support in various data management systems and proposed solutions to some of the important problems.

References

[1] K. Nahrstedt and R. Steinmetz. Resource Management in Networked Multimedia Systems. *IEEE Computer*, 28(5):52–63, 1995.

[2] D. Yau and S. Lam. Operating System Techniques for Distributed Multimedia. *International Journal of Intelligent Systems*, 13(12):1175–1200, 1998.

[3] Elisa Bertino, Ahmed Elmagarmid and Mohand-Saïd Hacid. A Database Approach to Quality of Service Specification in Video Databases. *SIGMOD Record*, 32(1):35–40, 2003.

[4] Y. Tu, S. Prabhakar and A. Elmagarmid. A Database-Centric Approach to Enabling End-to-End QoS in Multimedia Repositories. Technical Report CSD TR 03-031, Purdue University, 2003.

[5] D. Carney, U. Çetintemel, A. Rasin, S. Zdonik, M. Cherniack and M. Stonebraker. Operator Scheduling in a Data Stream Manager. In *Proc. of 29th VLDB Conf.*, pages 838–849, August 2003.

[6] Y. Tu, S. Prabhakar, A. Elmagarmid and R. Sion. QuaSAQ: An Approach to Enabling End-to-End QoS for Multimedia Databases. To appear in *Proc. of 9th EDBT Conf.*, 2004.

[7] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar and J. Hansen. A Scalable Solution to the Multi-Resource QoS Problem. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1999.

[8] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul and S. Zdonik. Monitoring Streams - A New Class of Data Management Applications. In *Proc. of 28th VLDB Conf.*, pages 84–89, August 2002.

[9] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack and M. Stonebraker. Load Shedding in a Data Stream Manager. In *Proc. of 29th VLDB Conf.*, pages 309–320, August 2003.

[10] D. L. Davison and G. Graefe", Dynamic Resource Brokering for Multi-User Query Execution. In *Proceedings of SIGMOD '95*, pages 281–92, June 1995.

[11] P. S. Yu and D. W. Cornell. Buffer Management Based on Return on Consumption In a Multi-Query Environment. *VLDB Journal*, 2(1):1–7, January 1993.