A Database-Centric Approach to Enabling End-to-End QoS for Multimedia Repositories

Yi-Cheng Tu, Sunil Prabhakar, Ahmed Elmagarmid
Department of Computer Sciences, Purdue University
West Lafayette, IN, 47907, USA
Email: {tuyc, sunil, ake}@cs.purdue.edu

ABSTRACT

The paper discusses the design and prototype implementation of a QoS aware multimedia database system. Recent research in multimedia databases has devoted little attention to the aspect of the integration of QoS support at the user level. One common scenario which we are concerned with connects a user through a visual interface to a multitude of media object stores. The user demands satisfiability of a set of quality parameter bounds specified at query time or before (via user profile mappings). The user is not aware of detailed low-level QoS parameters but rather specifies high-level, qualitative attributes on the media query. Our proposed architecture to enable end-to-end QoS control, the QoS-Aware Query Processor (QuaSAQ), is motivated by query processing and optimization techniques in traditional database management systems. The proposed solution relies on mediation through several components (two of which are the QoP Browser and Quality Manager) that enable searching, locating, composing and presenting of multimedia objects with associated QoS constraints. In addition to an overview of key research issues, this paper also presents some of the proposed design solutions. One focus problem is how to evaluate the alternative plans for serving QoS-enhanced queries. We propose a novel *cost model* that explicitly takes the resource utilization of plans and the current system contention level into account. Experiments run on the QuaSAQ prototype show significantly improved QoS and throughput in media query processing.

General Terms

QoS, multimedia database, video database

Keywords

plan generation, query optimization

1. INTRODUCTION

As compared to traditional applications, multimedia applications have special requirements with respect to search and playback with satisfactory quality. The problem of searching multimedia data has received significant attention from researchers with the resulting development of content-based retrieval for multimedia databases. The problem of efficient retrieval and playback of such data (especially video data), on the other hand, has not received the same level of attention. From the point of view of multimedia database design, one has to be concerned about not only the *correctness* of query results, but the *quality* of the media objects (as part of the query result) delivered to the users as well. The set of quality parameters that describes the temporal/spatial guarantees of media-related applications is called *Quality of Service* (QoS) [26]. Guaranteeing QoS for the user requires an end-to-end solution – all the way from the retrieval of data at the source to the playback of the data at the user. The Internet and the World Wide Web, both currently based on quality over-provisioning schemes, shows QoS provisioning is not the main focus in network design. Similarly, operating system (OS) support of QoS is not readily available in most end-point systems. Nevertheless, quality critical, real-time multimedia applications cannot be built on top of plain TCP/IP network and general time-sharing OS.

In spite of the fact that research in multimedia databases has covered many key issues such as data models, system architectures, query languages, algorithms for effective data organization and retrieval [11], little effort has been devoted to the aspect of the integration of QoS support at the higher levels. In the context of general multimedia system, research on QoS has concentrated on system and network support with little concern for QoS control on the higher (user, application) levels. High-level QoS support is essential in any multimedia systems because the satisfaction of human users is the primary concern in defining QoS [2]. Simply deploying a multimedia database system on top of a QoS-provisioning system will not provide end-to-end user-level QoS. QoS specification and mapping on the user and application levels are domain-specific problems. Therefore, current solutions on the low-level QoS are not enough to meet the challenges faced by individual applications. Moreover, such a solution is unable to exploit the application level flexibility such as the user's acceptable range of quality, or notion of correct results. For example, for a physician diagnosing a patient, the jitter-free playback of very high frame rate and resolution video of the patient's test data is critical; whereas a nurse accessing the same data for organization purposes may not require the same high quality. Such information is only available at the user or application levels.

We envision users such as medical professionals accessing these databases via a simple user interface. In addition to specifying the multimedia items of interest (directly or

Technical Report CSD-TR-0331, Department of Computer Sciences, Purdue University, 2005

via content-based similarity to other items), the user specifies a set of desired quality parameter bounds. The quality bounds could be specified explicitly or automatically generated based upon the user's profile. The user should not need to be aware of detailed system QoS parameters but rather specifies high-level qualitative attributes (e.g. "high resolution", or "CD quality audio"). Part of our research is related to translating high-level user queries or actions into queries with associated QoS and security parameters, unique to each user. Thus a QoS-enabled database will search for multimedia objects that satisfy the content component of the query and at the same time can be delivered to the user with the desired level of quality.

In this paper we discuss the design and prototype implementation of QuaSAQ – our QoS-aware multimedia database system. We describe the major challenges to enabling end-to-end QoS, and present our proposed solutions to these problems. To the best of our knowledge, end-to-end QoS has never been achieved in any other prototype system for multimedia databases. We present experimental results from our prototype system that establish the feasibility and advantages of such a system for guaranteeing user-level QoS. Our implementation builds upon the VDBMS prototype multimedia database system developed by our group at Purdue University [1]. VDBMS provides a rich set of mechanisms to retrieve image and video data by content. It is built upon the open-source Predator database system and the SHORE storage manager. Among other enhancements, QuaSAQ extends VDBMS to build a distributed QoS-aware multimedia DBMS with multiple copies of storage/streaming manager. Our work builds upon system and network QoS provisioning tools such as QualMan [17] and GARA [7].

To address the structure of a QoS-provisioning networked multimedia system (NMS), four levels of QoS have been proposed: user QoS, application QoS, system QoS and network QoS [26, 19]. The specifications and semantics of QoS on these levels are different. To achieve end-to-end control of QoS in such heterogeneous, dynamic environments, all components in the NMS must cooperate to provide real-time service. These components include software and hardware entities in the end systems as well as network routers along the data delivery path. As the starting point of QoS control, user and application QoS interact with the lower levels for resource guarantees and status feedback. One type of such interactions is *QoS translation* or *QoS mapping* that is generally done from high levels to low levels. QoS mapping from low levels to high levels is either useless or infeasible. Thus, QoS on system and network levels are internal entities that users never need to be aware of.

Although an agreement on the set of most relevant QoS parameters in multimedia databases is yet to be reached among researchers, we consider a series of QoS parameters in our research as shown in Table 1. User-level QoS is not included in Table 1 since they are basically abstractions of application QoS (Section 3.2.1).

QoS guarantees for individual requests and the overall system performance are in most cases two conflicting goals since the entire QoS problem is caused by scarcity of resources. To approach this problem, we need to be able to identify bottleneck resources and generate alternative execution strategies with different resource consumption features for each request. Most of the research on QoS fails to address the optimization of system performance. In this paper,

Table 1: Examples of QoS parameters in video databases.

QoS Level	QoS Parameter
Application	*Frame Width, Frame Height, Color Resolution, Time Guarantee, Signal-to-noise ratio (SNR), Security*
System	*CPU cycles, Memory buffer, Disk space, and bandwidth*
Network	*Delay, Jitter, Reliability, Packet loss, Network Topology, Bandwidth*

we highlight the key elements of our proposed approach to supporting end-to-end QoS and achieving high performance in a multimedia database environment. The approach is motivated by query processing and optimization techniques in conventional distributed databases.

The key idea of our approach is to augment the query evaluation and optimization modules of a distributed database system (D-DBMS) to directly take QoS into account. To incorporate QoS control into the database, user-level QoS parameters are translated into application QoS and become an augmented component of the query. For each raw media object, a number of copies with different application QoS parameters are generated offline by transcoding and these copies are replicated on the distributed servers. Based on the information of data replication and possible QoS adaptation options (e.g. frame dropping during playback), the query processor generates various plans for each query and evaluates them according to a predefined *cost model*. The query evaluation/optimization module also takes care of resource reservation to satisfy low-level QoS. For this part, we propose the design of a unified API and implementation modules, that enable negotiation and control of the underlying system and network QoS APIs, thereby providing a single entry-point to a multitude of QoS layers (system and network).

The proposed solution is implemented using several components that enable searching, locating, composing and presenting multimedia objects with associated QoS constraints. Each aspect of the overall process involves a set of complex issues and the interaction between the components is particularly challenging. We present the most important ones together with envisioned design solutions in this paper.

The major contributions of this paper are: 1) We propose a query processing architecture for multimedia databases for handling queries enhanced with QoS parameters; 2) We propose a cost model that evaluates QoS-aware queries by their resource utilization with consideration of the current system status. To the best of our knowledge, this is the first such effort; The cost model can be generalized into a broader range of scenarios in the area of distributed systems; and 3) We implement the proposed query processor within a multimedia DBMS and evaluate our design via experiments run on this prototype.

The paper is organized as follows. Section 2 deals with the main issues encountered in the process of designing and implementing the system. Section 3 presents the actual architecture of the Quality of Service Aware Query Processor (QuaSAQ). We also discuss details pertaining to the design of individual components in the architecture. The prototype implementation of QuaSAQ is detailed in Section 4. Section 5 presents the evaluation of the proposed QuaSAQ architecture. In Section 6, we compare our work with relevant research efforts. Section 7 concludes the paper.

2. ISSUES

Building a distributed multimedia database system requires a careful design of many complex modules as well as effective interactions between these components. This becomes further complicated if the system is to support non-trivial aspects such as QoS. Important research issues include: identifying and defining an acceptable set of user-level quality parameters that match the requirements of real-time distributed multimedia applications; mapping those parameters to lower-level definitions and enabling user control in their specification; the plan generation for multimedia presentations; evaluation of the costs of generated plans; provisioning the quality at all the underlying levels according to high level user specs; displaying/presenting the multimedia objects satisfying all QoS bounds.

Conceptually, a D-DBMS accepts a query over relations that are distributed across multiple sites and translates it into a sequence of operations to be performed at individual sites so as to compute the answer to the query. This translation is based upon metadata that describes the characteristics of the data, e.g. distribution, replication, existence of indices, etc. The primary criterion for identifying the best plan from among available alternatives is the cost estimate of each plan. In order to extend this D-DBMS approach to address the end-to-end QoS problem, several important requirements have to be met. These include:

1. Smart QoS-aware data replication algorithms have to be developed. Individual multimedia objects need to be replicated on various nodes of the database. Each replica may satisfy different application QoS in order to closely meet the requirements of user inputs. In other words, we trade storage space for runtime QoS-related media transformation cost (e.g. transcoding). The total number and choice of QoS of pre-stored media replicas should reflect the access pattern of media content. Therefore, dynamic online replication and migration has to be performed to make the system converge to the current status of user requests. Another concern in replication is storage space. Ideally, the relative storage used for replication to that used for the original media should be bounded by a constant.

2. Mapping of QoS parameters between different layers has to be achieved. First of all, user-level qualitative QoS inputs (e.g. DVD-quality video) need to be translated into application QoS (e.g. spatial resolution) since the underlying query processor only understands the latter. One critical point here is that the mapping from user QoS to application QoS highly depends on the user's personal preference. This problem is likely to be solved by the use of *user profiles* [8, 27]. Resource consumption of query plans is essential for cost estimation and query optimization in QoS-aware multimedia databases. This requires mapping application QoS in our QoS-enhanced queries to QoS parameters on the system and network level, which is a difficult problem considering the dynamic system content and heterogeneous platform configurations.

3. The *search space* of possible execution plans in the QoS-aware multimedia database is of a very different structure from that of a traditional D-DBMS. In the latter, the primary data model for search space comprises a *query tree* due to the dominant cost of performing joins. The query optimizer then explores the space using strategies such as *dynamic programming* and *randomized search* to find the "best" plan according to a *cost model* [21]. In our system, various components such as encryption, encoding, and filtering must be individually considered in addition to the choice of database server and physical media object. Depending on the system status, any of the above components can be the dominant factor in terms of cost. In the context of our QoS-aware database, novel data structures must be built to effectively model the search space and process plans in the space. Specifically, the query generator should be able to efficiently *prune* the plans that do not meet the user QoS requirements and traverse the space for finding optimal plans.

4. A cost estimation model is needed to evaluate the generated QoS-aware plans. Unlike the static cost estimates in traditional D-DBMS, it is critical that the costs under current system status (e.g. based upon current load on a link) be factored into the choice of an acceptable plan. Furthermore, the cost model in our query processor should also consider optimization criteria other than the *total time*[1], which is normally the only metric used in D-DBMS. A very important optimization goal in multimedia applications is system throughput. Resource consumption of each query has to be estimated and controlled for the system to achieve maximum throughput and yet QoS constraints of individual requests are not violated. If we consider other factors such as real-world cost (monetary price) of the query plans, the cost model becomes more complicated.

5. Once an acceptable quality plan has been chosen, the playback of the media objects in accordance with the required quality has to be achieved. Generally, QoS control in multimedia systems are achieved in two ways: *resource reservation* and *adaptation* [26]. Both strategies require deployment of a QoS-aware resource management module, which is featured with *admission control* and *reservation* mechanisms. There may also be need for *renegotiation* (*adaptation*) of the QoS constraints due to user actions during playback.

Our research on QoS-aware databases is an attempt to address all the above challenges. In the next section, we present a framework for QoS provisioning in a distributed multimedia database environment with the focus on our solutions to items 3 and 4 listed above. For items 1, 2 and 5, we concentrate on the implementation and evaluation of known approaches within the context of multimedia databases. We also propose new mechanisms to improve the current solutions.

3. QUALITY-OF-SERVICE AWARE QUERY PROCESSOR (QUASAQ)

[1]Sometimes *response time* is also used, as in distributed INGRES.

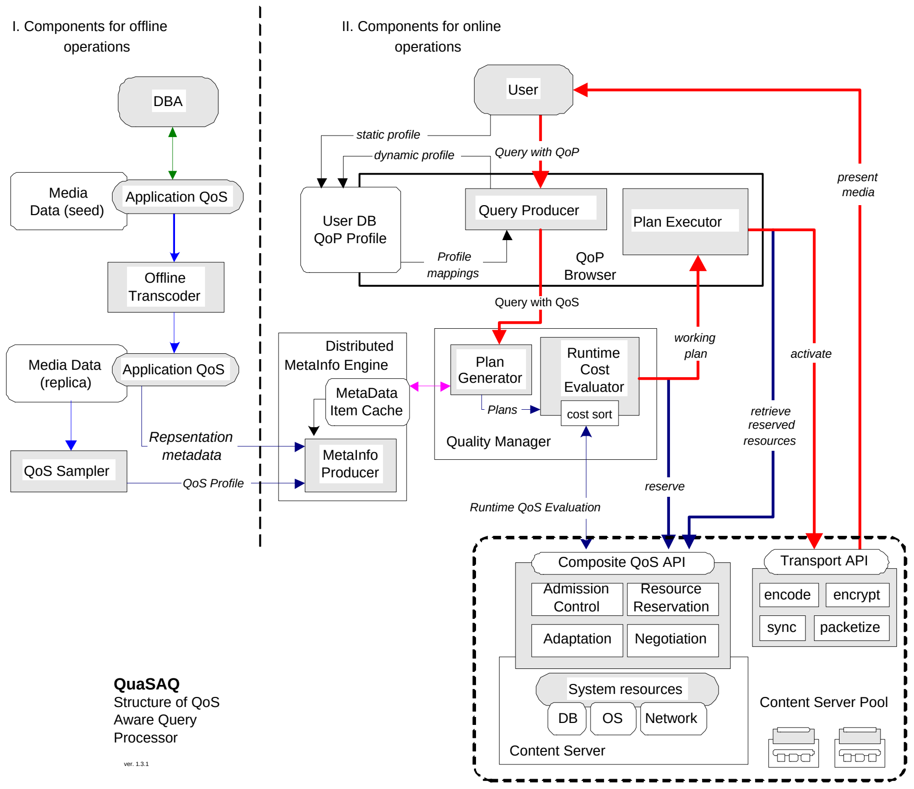


Figure 1: QuaSAQ architecture.

Figure 1 describes in detail the proposed architecture of our QoS-aware distributed multimedia DBMS, which we call Quality-of-Service Aware Query Processor (QuaSAQ). In this section, we present detailed descriptions of the various components of QuaSAQ.

Supporting QoS in the database framework will be enabled through the creation of the following major components: a *transcoder* that generates replicas from the seed copies of media objects; a offline *QoS Sampler* that performs static QoS mapping; a *QoP Browser* front end that appends quality information to user queries; an enhanced *Metadata Engine* that contains media content as well as QoS related metadata; a *Plan Generator* that creates alternative execution plans based upon the user query augmented with QoS, it also annotates the plan with low-level constraints for each component based upon a translation of the user's QoS requirements; a *runtime cost evaluator* that calculates the cost of the annotated plans based on a specific *cost model*; a *Composite QoS API* that interfaces to each of the underlying components to determine the current state of the system, request, and reserve resources on behalf of plans; a *transport API* and an *plan executor* that together display the results to the user using the resources that have been reserved for the query plan.

3.1 Offline Components

The offline components of QuaSAQ provide a basis for the database administrators to accomplish QoS-related database maintenance. Two major activities, *offline replication* and *QoS sampling*, are performed for each media object inserted into the database. As a result of offline activities, relevant information such as the quality, location and resource consumption pattern of each replica of the newly-inserted object is fed into the *Distributed Metadata Engine* and stored as metadata.

3.1.1 Offline replication

The primary function of the offline components is to replicate the original media data on the distributed database servers. Unlike normal data replication operations, each replica in QuaSAQ has different application-level QoS parameters. We call this *QoS-specific replication* and it is accomplished by the *offline transcoder*. The use of replicas provides flexibility in the QoS-enabled multimedia retrieval because the system can choose the copy with QoS that are the closest to user requirements. It is also called *static adaptation* in general multimedia system jargon. The word "static" means that the adaptation is made before the streaming session begins. An alternative to static adaptation would be transcoding the raw media on-the-fly, which may be constrained by resource availability at runtime.

Although storage space is usually not a bottleneck resource in a typical media streaming system, the extra disk space used for static adaptation should not grow unboundedly. Apparently, there is an tradeoff between availability and storage use. From empirical equations we derived to estimate the storage consumption of QoS-specific replication (Section 3.1.3), we can estimate the storage occupation for QoS-specific replication.

According to Figure 3a, the relative bitrate of a video replica with one single reduced QoS parameter (e.g. spatial resolution, temporal resolution, and signal-to-noise ratio (SNR)) is expressed as:

$$B = aS^3 + bS^2 + cS + d \tag{1}$$

where B_0 is the ratio of the bitrate of the generated video over that of the original video, S is the ratio of change in QoS ($0 \leq S \leq 1$), and a, b, c, d are constants derived from experiments. Suppose we replicate a media into N copies by changing one QoS parameter (whose value falls into a domain of range P), each of which has its S value. Without any knowledge of the distribution of S in the queries that access the media object, we can choose a series of QoS degradation percentage values S_i $(i = 1, 2, \ldots, N)$ that cover the domain of S uniformly, e.g. $S_i = \frac{i}{N}$. This limits the difference between the user-required QoS and that of any replicas to be smaller than $\frac{P}{N}$. This difference can be resolved either by renegotiation (Section 3.2.1) or online transcoding (Section 3.4.1). The sum of the relative bitrate of all such replicas can be obtained by:

$$\begin{aligned}\sum_{i=1}^{N-1} B_i &= \sum_{i=1}^{N-1} \left(aS_i^3 + bS_i^2 + cS_i + d\right) \\ &= \sum_{i=1}^{N-1} a\left(\frac{i}{N}\right)^3 + b\left(\frac{i}{N}\right)^2 + c\frac{i}{N} + d,\end{aligned}$$

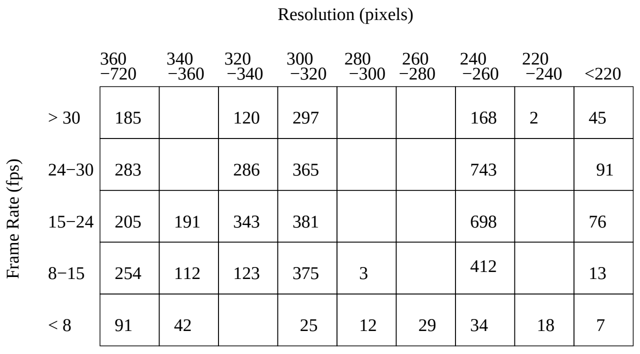
where B_i is the bitrate for the i-th replica. The storage occupation can be easily calculated as $T\sum_{i=1}^{N-1} B_i$ where T is the playback time of the media. It is not hard to see that the total relative storage use is $O(N)$. In other words, the storage needed for replication along one quality dimension is linearly related to the number of copies made. Considering the existence of multiple QoS parameters, the situation could be much worse. Only when N is a small number can we perform QoS-specific replication.

Since it is not affordable to produce more than a (small) fixed number of copies for each object, the choice of quality parameters of these limited number of copies becomes important. The main idea for solving this decision-making problem is that the replication should reflect the access frequency of individual combinations of QoS values. When the media object is first inserted into the database, we can replicate it to copies with a set of "standard" QoS according to some convention (Table 2). For example, a video with resolution of 352×240, 24bit color, and frame rate of 23.97fps is often regarded as of "VCD quality". An assumption here is the tendency for people to ask for videos with such QoS.

3.1.2 Online replication

The initial set of replicas may prove to be the wrong guess as time goes by. We need to modify the composition of the QoS-specific replicas as we gain more knowledge on the access pattern of queries on the domain of QoS parameters. This is called *Dynamic replication*. Unlike traditional data replication problem that only considers the availability of the content of the data, the QoS-specific replication in multimedia database has to consider the availability of the quality of the data. An intensive study on QoS-specific replication will be presented in a forthcoming paper. In this paper, we only briefly introduce a two-level replication algorithm.

The algorithm consists of 2 stages: a *content replication* stage and a *quality replication* stage. On the first stage,



Frame Rate (fps) \ Resolution (pixels)	360–720	340–360	320–340	300–320	280–300	260–280	240–260	220–240	<220
> 30	185		120	297			168	2	45
24–30	283		286	365			743		91
15–24	205	191	343	381			698		76
8–15	254	112	123	375	3		412		13
< 8	91	42		25	12	29	34	18	7

Figure 2: An illustrative 2-D QoS frequency table.

we use a threshold-based algorithm to determine the storage allocated to a specific media object, as described in [**?**]. When the availability of a certain media object is below the threshold, replication is triggered. The decision of what quality replica to be produced is made on the second stage of the algorithm. For this purpose, QuaSAQ needs to keep an access frequency table. The table contains QoS sets requested as well as the frequency of these QoS sets (starting from some checkpoint). The space of all possible QoS sets with n QoS parameters can be viewed as a n-dimensional hypercube, in which each dimension can be divided into a finite number of regions (assuming each QoS parameter has finite number of discrete values). Figure 2 shows such a space composed of 2 QoS parameters: resolution and color depth. The cells in the hypercube are filled with frequencies of requests to the QoS sets represented by the cell. We put these frequencies in a sorted list. When a replication is needed, we retrieve the cell from the head of the list and replicate according to the QoS referred to by the cell.

3.1.3 QoS mapping

Translation of QoS between neighboring levels is an essential component of QoS-aware databases. In QuaSAQ, we are mainly interested in the QoS translation from user level to application level and application level to system level. The QoS mapping between system and network levels is handled by a *Composite QoS API* that takes care of QoS-related resource management and scheduling (Section 3.5). The mapping of user QoS to application QoS is accomplished by user-specific profiles, as described in Section 3.2. This leaves the translation of application QoS to system (resource) QoS, which can be denoted as

$$\vec{Q} = \{q_1, q_2, \ldots, q_m\} \longrightarrow \vec{R} = \{r_1, r_2, \ldots, r_n\}$$

if we consider m application QoS parameters and n system resources. In other words, the purpose of the mapping is to generate a *resource vector* $\vec{R}$ from a series of application QoS inputs $\vec{Q}$. This is a non-trivial problem due to the dynamically changing system load and heterogeneous platform architecture.

There are two major QoS mapping strategies [27]: a static table-driven method and a statistical sampling method. The first solution involves building a table with different combinations of application QoS and their relevant resource use while mapping is performed by table lookups. The second attempts to compose a mathematical function that handles

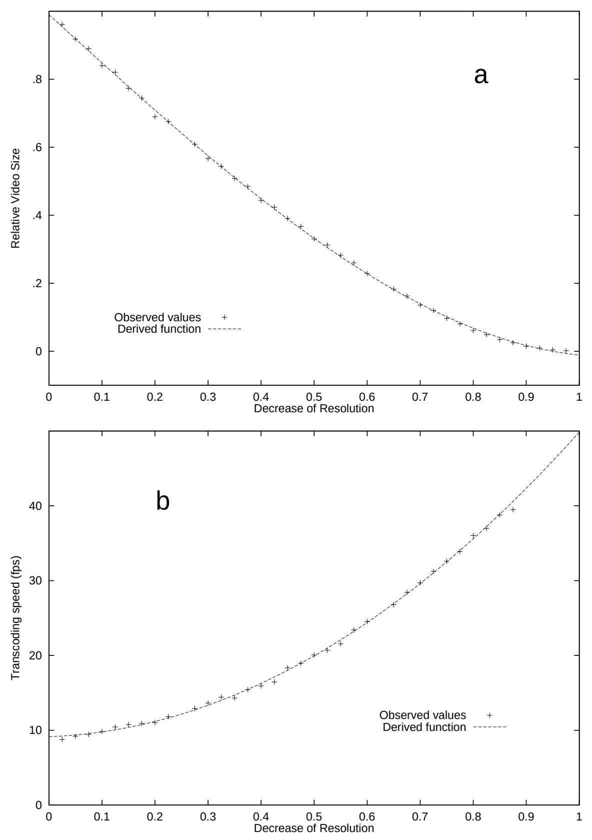


Figure 3: Functions and experimental results for mapping between spatial resolution to resources. a. Bandwidth; b. CPU use.

mapping of all the possible values in the range of QoS parameters based on a number of mapping samples ($\vec{Q}, \vec{R}$ pairs) collected. The latter is more adaptive to QoS changes but is less precise due to variations. One of the tools for the sampling method are Spline functions [27]. In QuaSAQ, we use an algorithm that takes advantages of both methods. When a new media and all its QoS-specific replicas are inserted into the database, the *QoS sampler* records the resource vector of these physical media by running a few iterations of media playback. These data serve as entries of a *QoS mapping table* as well as samples for computing a general mapping function. This leads to a two-step QoS mapping procedure at runtime: we first look up the QoS mapping table; if there is no matching entry in the table, we then use the mapping function to calculate the resource vector. Both QoS mapping table and the mapping functions are stored as metadata associated with each media object.

Translation of application QoS to different resources has to be considered individually. The network bandwidth is given by the bitrate of the media retrieved or the expected bitrate of the media (generated by online transcoding) from mapping functions. For the CPU cycles, we define a *Period* and *Peak Processing Time* (PPT) for each job, as described in [5]. For continuous media applications, the *Period* is generally the reciprocal of the number of frames per second while the PPT is the CPU time needed within each period. The CPU use of a media delivery job can thus be conveniently expressed as $\frac{PPT}{Period}$. Two of the mapping functions we derived are shown in Figure 3. The left graph represents the mapping from various resolution (other QoS unchanged) to the bandwidth (bitrate) while the right graph shows resolution to CPU usage. Both mappings are solved as a polynomial function. The function for the left graph is $y = 0.4212x^3 - 0.0031x^2 - 1.4191x + 0.9898$ while that for the right graph is $y = 38.1841x^2 + 2.5341x + 9.1419$. With reservation-based QoS resource management, these functions are found to be very accurate.

3.2 QoP Browser

The QoP Browser is the user interface to the underlying storage, processing and retrieval system. It enables certain QoP parameter control, generation of QoS-aware queries, and execution of the resulting presentation plans. The main entities of the QoP Browser include:

- The *User Profile* contains high-level QoP parameter mappings to lower level QoS parameter settings as well as various user related statistics acquired over time, enabling better renegotiation decisions in case of resource failure.
- The *Query Producer* takes as input some user actions (requests with QoP inputs) and the current settings from the user profile and generates a query. As compared to those of traditional DBMS, the queries generated in QuaSAQ are enhanced with QoS requirements. We call them *QoS-aware queries*.
- The *Plan Executor* is in charge of actually running the chosen plan, after initial QoS provisioning has taken place. It basically performs actual presentation, synchronization as well as run-time renegotiation of underlying QoS parameters.

3.2.1 Quality of Presentation

From a user's perspective, QoS translates into the more qualitative notion of *Quality of Presentation* (QoP). The user is not expected to understand low level quality parameters such as frame rates or packet loss rate. Instead, the user specifies high-level qualitative parameters to the best of his/her understanding of QoS. Some key QoP parameters that are often considered in multimedia systems include: spatial resolution, temporal resolution or period, color depth, reliability, audio quality, and monetary costs. Before being integrated into a database query,the QoP inputs are translated into application QoS based on the information stored in the *User Profile*. For example, a user input of "VCD-like spatial resolution" can be interpreted as a resolution range of 320×240 – 352×288 pixels. The application QoS parameters are quantitative and we achieve some flexibility by allowing one QoP mapped to a range of QoS values. Furthermore, the design of User Profile allows for dynamical updates at runtime as well as through the use of a statistical model, trained by live data.

A distinction is needed to differentiate between static and dynamic QoP provisioning. In the first case, quality requirements are assumed to be stable in time and, in most cases, no renegotiation process is needed. In the dynamic case, QoS requirements can be modified and a renegotiation is expected. Our focus is on mechanisms that support static

QoP but can be extended to support dynamic QoS. Another scenario for renegotiation is when the user-specified QoP is rejected by the admission control module due to low resource availability. Under such circumstances, a number of admittable alternative plans will be presented as a "second chance" for the query to be serviced.

Most of the definitions that have been proposed to the notion of QoP in continuous media, including the one used in our QuaSAQ design, are qualitative. One important weakness of these qualitative formulations of QoP is their lack of flexibility (i.e. low level processing does not take into account differences between users). For example, when renegotiation has to be performed, one user may prefer reduction in the temporal resolution while another user may prefer a reduction in the spatial resolution. We remedy this by introducing a *per-user weighting* of the quality parameters as part of the User Profile. This process delivers additional expressive power in specifying parameters. In particular, the weighting is also used in the renegotiation process. Given the fact that weights are user defined as well as statistically adjusted over time, this can be thought of as "user-tailored" renegotiation.

On the application level, user-specific weighting can be used to calculate the *utility* of execution plans for QoS-aware queries. Utility, a term first used in economics, is a measure of human preference on alternatives towards the same goal [16]. In the context of multimedia systems, utility can be viewed as the user's satisfiability of the media he/she received. The utility of an alternative is generally expressed as a real number through a *utility function*. In our QoS-aware multimedia DBMS, the following utility function can be used to capture a user's preference on an alternative query plan:

$$U(\vec{Q}) = \sum_{i=1}^{n} W_i f_i(Q_i) \tag{2}$$

where $\vec{Q} = q_1, q_2, \ldots, q_n$ is a vector containing the values of all n QoS parameters, W_i is the user-specific weight ($\sum_{i=1}^{n} W_i = 1$), and f_i the utility function of a single QoS parameter i. There are two ways to obtain the per-parameter utility function f_i: it can be an input from individual users or from real world experiments on people's perception of media quality [6]. In QuaSAQ, the utility of QoS sets is an important factor in query optimization (Section 3.4).

3.2.2 The query language.

Integrating QoS within a querying system also implies query language level extensions in syntax as well as in semantics. One choice is an extended version of SQL. An example of such an extension may be a new QUALITY clause that a user can add to each query. This clause captures any specific quality requirements that should be maintained when answering the query. One draft illustrative example is:

```
SELECT vid:[s,e]
FROM video:VidLib1
WHERE (vid, s, e) IN FindVideoWithObject( Someone )
QUALITY Resolution = High, Delay = Low
```

3.3 Distributed Metadata Engine

Metadata are descriptions of raw data items stored in a database for the purposes of faster access, better manageability and shareability of large sets of structured and/or unstructured data [10]. In a multimedia DBMS, operations such as content-based searching depend heavily, if not exclusively, on the metadata of the media objects. As mentioned in Section 2, we assume a distributed environment in which video objects are stored in several locations. Each document can have one or more instances and sub-components, each with different representation characteristics. This results in more items in the metadata collection. Specifically, we require at least the following types of metadata for a QoS-enabled DBMS:

- *Content Metadata:* describe the content of objects to enable multimedia query, search, and retrieval. In QuaSAQ, a number of visual and semantic descriptors such as shot detection, frame extraction, segmentation, and camera motion are extracted. For simplicity, the content metadata of only one instance of each video is extracted and stored in QuaSAQ.

- *Quality Metadata:* describe the quality characteristics (in the form of application level QoS) of physical media objects. For our QoS-aware DBMS, the following parameters are kept as metadata for each video object: resolution, color depth, frame rate, and file format.

- *Distribution Metadata:* describe the physical locations (i.e. paths, servers, proxies, etc.) of the media objects. This includes object identifications (OIDs) of all replicas of the same media. It records the OIDs of objects and the mapping between media content (logical object) and media file (physical object).

- *QoS profile:* describe the resource consumption in the delivery of individual media objects. The data in QoS profiles is obtained via static QoS mapping performed by the *QoS sampler*. The QoS profiles are the basis for cost estimation of QoS-aware query execution plans.

The Quality metadata and Distribution metadata will be used in *Quality Manager* to identify and evaluate alternative plans as well as initial translation of user-level QoP parameters to low-level QoS parameters. An important design issue is related to distributing vs. centralizing metadata. We distribute the metadata in locations that are close to the actual objects enabling ease of use and migration. Caching is used to accelerate non-local metadata accesses.

3.4 Quality Manager

The Quality Manager is the focal point of the entire system. It is heavily integrated with the *Composite QoS APIs* in order to enable reservation and negotiation. It has the following main components:

3.4.1 Plan Generator

The *Plan Generator* is in charge of generating plans that enable the execution of the query from the Query Producer. It uses metadata retrieved from the Distributed Metadata Engine in order to locate suitable target objects for each query.

The Content Metadata is used to identify logical objects that satisfy the content component of the query (e.g. videos with images of George Bush or Sunsets). As described in Section 3.1.1, a given logical object may be replicated at multiple sites and further with different QoS. The plan generator determines which of the alternatives can be used to

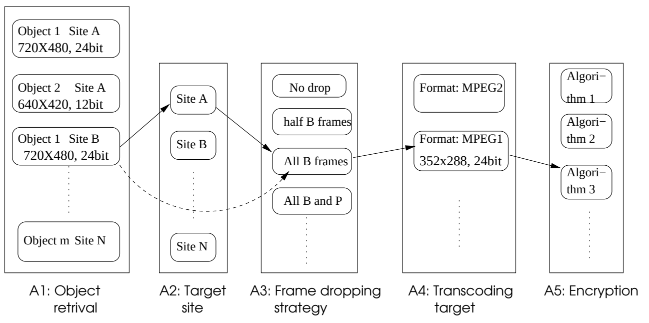


Figure 4: *Illustrative plan generation in QuaSAQ.*

satisfy the request and also the necessary steps needed to present it to the user. The necessary information is stored as Quality Metadata and Distribution Metadata.

The final execution of QoS-aware query plans can be viewed as a series of *server activities* that may include retrieval, decoding, transcoding between different formats and/or qualities, and encryption (we treat security as a form of quality). Therefore, the search space of alternative QoS-aware plans consists of all possible combinations of media repositories, target objects, and server activities mentioned above. We can model the search space as a universe of disjoint sets. Each set represents a target media object or a server activity whose possible choices serve as elements in the set. Suppose we have n such sets $A_1, A_2, \ldots, A_n$, then an execution plan is an ordered set $a_1, a_2, \ldots, a_m$ satisfying the following conditions:

(1) $m \leq n$;
(2) $\forall a_i \ (1 \leq i \leq m), \exists \ A_j \ni a_i \ (1 \leq i \leq n)$;
(3) For any $i \neq j$ with $a_i \in A_k$ and $a_j \in A_l$, we have $k \neq l$.

The semantics of the above conditions are: (1) The total number of components in a plan cannot exceed the number of possible server activities; (2) All components in a plan come from some disjoint set; and (3) No two components in a plan come from the same set. The size of the search space is huge even with the above restrictions. Suppose each set of server activity has d elements, the number of possible plans is $O\big(n!d^n\big)$. Fortunately, there are also some other system-specific rules that further reduce the number of alternative plans. One salient rule relates to the order of server activities. For example, the first server activity should always be the retrieval of a media object from a certain site, all other activities such as transcoding, encryption have to follow the the media retrieval in a plan. If the order of all server activity sets are fixed, the size of search space decreases to $O\big(2^n d^n\big)$.

Runtime QoS Evaluation and Plan Drop. The Plan Generator described above does not check generated plans for any QoS constraints but rather annotates the plan with an initial translation of QoP. We can perform those verifications by applying a set of static and dynamic rules. First of all, decisions can be made instantly based on QoS in the query. For example, we cannot retrieve a video with resolution lower than that required by the user. Similarly, it makes no sense to transcode from low resolution to high resolution. It is easy to see that QoS constraints help further reduce the size of search space by decreasing the appropriate set size d. In practice, d can be regarded as a constant. Some of the plans can be immediately dropped by the *Plan Generator* if their costs are intolerably high. This requires QuaSAQ to be aware of some obvious performance pitfalls. For example, encryption should always follow the frame dropping since it is a waste of CPU cycles to encrypt the data in frames that will be dropped. Once a suitable plan has been discovered, the Plan Generator computes its resource vector and feeds it to the next component down the processing pipe-line (*Runtime Cost Evaluator*) until no more satisfying plans can be generated. The reduced size of search space makes the enumeration and evaluation of all candidate plans feasible (Section 5.2). In our experiments on a QuaSAQ prototype, we generally evaluate 30 - 40 plans for each query.

Illustrative examples of plans. The path in solid lines shown in Figure 4 represents a query plan with the following details: 1. retrieve physical copy number 1 of the requested media from the disk of server B; 2. transfer the media to server A; 3. drop all the B frames; 4. transcode to MPEG1 format with certain target QoS; 5. encrypt the media data using cipher 3. The dotted line corresponds to a simpler plan: retrieve the same object and drop B frames, no transcoding or encryption is needed. An even simpler plan would be a single node in set $A1$, meaning the object is sent without further processing.

3.4.2 Runtime Cost Evaluator

The *Runtime Cost Evaluator* is the main component that computes (at runtime) estimated costs for generated plans. It sorts the plans in ascending cost order and passes them to the Plan Executor in the QoP Browser. The earliest plan in this order that satisfies the QoS requirements will be used to service the query. In a traditional D-DBMS, the cost of a query is generally expressed as the sum of time spent on CPU, I/O and data transferring. In other words, the total/response time is used as the only metric in cost estimation. There are some major drawbacks in using this model in the QoS-aware multimedia database. First of all, it fails to consider the current status of the system resources thus the cost prediction is imprecise. Secondly, the only optimization is to minimize the number of I/Os (or the communication cost in a distributed DBMS) in plan execution under such a model. Other costs such as the shipping of query results to client are ignored. This is correct for traditional databases since the size of the query results are the same, regardless of what plan is executed. However, this is not true for multimedia databases where media delivery is also considered a part of query processing. Finally, the search results (media objects) are sent to the users in a streaming manner over a significant amount of time in multimedia databases. The total time for executing any query plans is exactly the same since the streaming time for a media object is fixed [2]. As a result, processing time is no longer a valid metric for cost estimation of the QoS-aware query plans.

In order to overcome the above problems, we propose a cost model that focuses on the resource consumption of alternative query execution plans. Multimedia delivery is generally resource intensive, especially on the network band-

[2]One exception is that the media can be downloaded in shorter time given enough bandwidth. We may also consider this as an option in QuaSAQ. However, media downloading has limited use in multimedia systems due to its high bandwidth consumption and limited buffer on the client side.

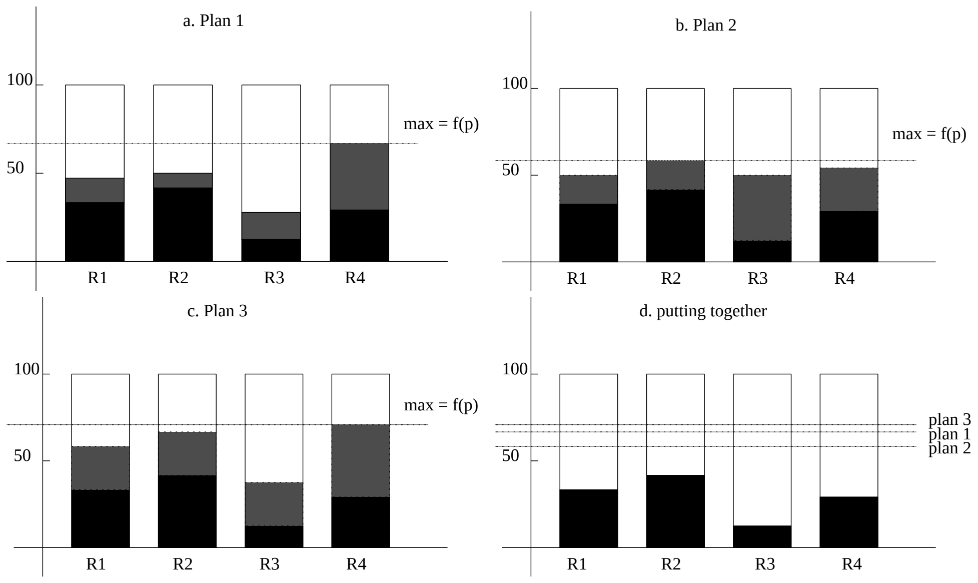


Figure 5: Cost evaluation by the Lowest Resource Bucket model.

width. Thus, to improve system throughput is an important design goal of media systems. Intuitively, the execution plan we may choose should be one that consumes as few resources as possible and yet meets all the QoS requirements. Our cost model is designed to capture the 'amount' of resources used in each plan. The central part of any cost model is a *cost function* that maps a plan to a real number. Because there are various types of system resources to consider and the relative importance of these resources are unknown, the development of the cost function based on resource utilization is not straightforward. Furthermore, the cost model should also be valid for other global optimization goals such as minimal waste of resources, maximized user satisfaction, and fairness. Our ultimate goal is to build a configurable query optimizer whose optimization goal can be configured according to user (DBA) inputs. We then evaluate plans by their *cost efficiency* that can be denoted as:

$$E = \frac{G}{C(\vec{R})}$$

where C is the cost function, $\vec{R}$ the resource vector of the plan being evaluated, and G the *gain* of servicing the query following the plan of interest. An optimal plan is the one with the highest cost efficiency. The generation of the G value of a plan depends on the optimization goal used. For instance, a *utility function* can be used when our goal is to maximize the satisfiability of user perception of media streams [25]. A detailed discussion of the configurable cost model mentioned above is beyond the scope of this paper. In the following, we present a cost model that aims to maximize system throughput.

3.4.2.1 Lowest Resource Bucket (LRB) model..

Suppose there are n types of resources we consider in evaluation of alternative plans of the QoS-aware queries. The total amount of these n individual resources provided by the system is $R_1, R_2, \ldots, R_n$. In our algorithm, we build n virtual *resource buckets*, each of which holds an individual system resource. The total amount of resources are standardized into the height of the buckets. The latter is represented as a unitless quantity (e.g. percentage) and is the same for all buckets. The buckets are filled when their relevant resources are being used and drained when the resources are released. Therefore, the height of the filled part of any bucket i is the percentage of resource i that is being used. For example, the filled part of bucket $R2$ in Figure 5d has height 42, which means 42% of R_2 is occupied. The cost evaluation is done as follows: for any plan p, we first transform the items in p's resource vector into standardized heights related to the corresponding bucket (denoted as $r_1, r_2, \ldots, r_n$); we then fill the buckets accordingly using the transformed resource vector and record the largest height among all the buckets. The query that leads to the smallest such maximum bucket height wins. The buckets are drained to the original heights after each query is evaluated and will only be filled when a plan is chosen to execute. In Figure 5, the cost of three plans (a, b, c) are marked by dotted lines. Putting them all together, we found the filled height of plan 2 is the lowest and plans 2 is chosen for execution. Formally, the *cost function* used for the LRB model can be expressed as

$$f(r_1, r_2, \ldots, r_n) = \max_{i=1}^{n} \left\{ \frac{U_i + r_i}{R_i} \right\} \tag{3}$$

where U_i is the current usage of resource i, r_i is the amount of resource i required for the plan, and R_i is the total amount of resource i provided by the system. The input is the resource vector of the plan being evaluated.

The reasoning of the above algorithm is easy to understand: the goal is to make the filling rate of all the *buckets* distribute evenly. Since no queries can be served if we have an overflowing bucket, we should prevent any single bucket from growing faster than the others. This algorithm is not guaranteed to be optimal, it works fairly well, as shown by our experiments (Section 5.2).

3.5 QoS APIs

Proprietary QoS API awareness is not desired in higher level components. The *Composite QoS API* hides implementation and access details of underlying APIs (i.e. system and network) at the same time offering control to upper layers (e.g. *Plan Generator*). One other handy advantage of the unified API approach is the ability to easily manage and quickly implement it on top of various platforms without necessarily having all the other QoS components in place. The major functionality provided through the *Composite QoS API* is QoS-related resource management, which is generally accomplished in the following aspects: 1.*Admission control*, which determines whether a query/plan can be accepted under current system status. The *Plan Generator* uses this to identify the plans whose resource requirements exceed availability; 2.*Resource reservation:* an important strategy toward QoS control by guaranteeing resources needed during the lifetime of media delivery jobs; 3.*Renegotiation* (*adaptation*) that are mainly performed under two scenarios: a. evaluation and presentation of alternatives when the user QoS requirements cannot be satisfied; b. change of QoS during the playback of media due to the change of user inputs. The type of system resources controlled depends on available releases of individual resource managing software. In the most relevant QoS resource management studies [17, 7, 28], CPU, memory, disk, and network bandwidth were considered for QoS control.

Transport API. It is basically composed of the underlying packetization and synchronization mechanisms of continuous media, similar to those found in general media servers. The Transport API has to honor the full reservation of resources. This is done through interactions with the appropriate reservation APIs as a part of the Composite QoS API. To harness the QoS-enabled network as well as the same layer of intermediate proxies, real-time stream media protocols such as RTP are needed to carry the data load of the media. Upper level system components (e.g. the *Plan Executor*) have to find support for runtime parameter renegotiation in order to adapt to the current status of the whole system. This should also be accomplished by the Transport API. The interface to some of the other *server activities* such as encryption, transcoding, and filtering are also integrated into the Transport API.

4. QUASAQ IMPLEMENTATION

We implement a prototype of QuaSAQ on top of the Video Database Management System (VDBMS) developed at Purdue University [1]. Similar to VDBMS, the QuaSAQ development is done using C++ under the Solaris 2.6 environment. Figure 6 shows the architecture of VDBMS enhanced with the QuaSAQ prototype.

4.1 QuaSAQ and VDBMS

Developed from the object-relational database engine PREDATOR[3] with Shore[4] as the underlying storage manager, VDBMS is a multimedia DBMS that supports full-featured video operations (e.g. feature-extraction, streaming) and complex queries (e.g. content-based searching). The PREDATOR [23] code covers DBMS components from user interface to query processor and leaves concurrency and recovery issues to the underlying Shore libraries [3]. Most of the VDBMS development was done by adding features to PREDATOR. We extended the current version of VDBMS, which runs only on a single node, to a distributed version by realizing communication and data transferring functionalities among different sites.

[3]http://www.distlab.dk/predator
[4]http://www.cs.wisc.edu/shore

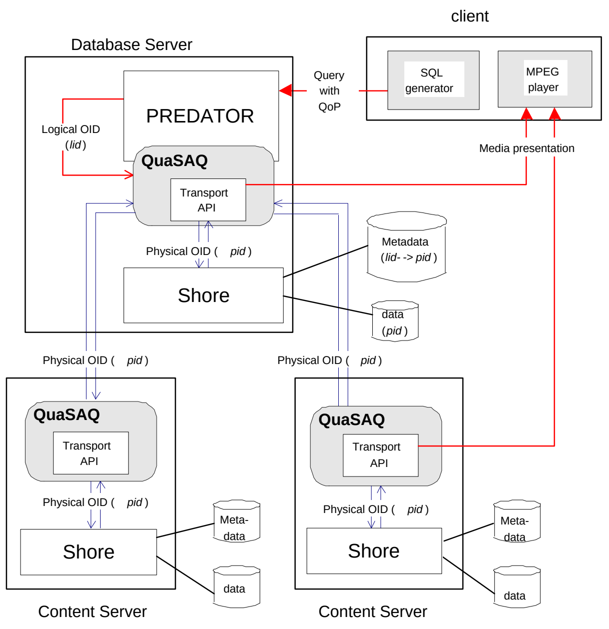


Figure 6: Architecture of QuaSAQ prototype.

As shown in Figure 6, QuaSAQ augments VDBMS and sits between Shore and PREDATOR in the query processing path. In our QuaSAQ-enhanced database, queries on videos are processed in two steps: 1. searching and identification of video objects done by the original VDBMS; 2. QoS-constrained delivery of the video by QuaSAQ [2]. To identify storage items (raw video, indices, relations, etc.) throughout the system, a 12-byte ID is assigned to each object according to the Shore convention. In the original VDBMS, the query processor returns such an object ID (OID), by which Shore retrieves the video from disk. With the consideration of QoS and replication of video data in QuaSAQ, the OIDs returned by the original query processor only refer to the video content but not the physical entity in storage since multiple copies of the same video exist. These OIDs are called the *logical OID* and those representing video replicas the *physical OID*. In QuaSAQ, the mapping between logical OIDs and physical OIDs are stored as part of the metadata (Section 3.3). Upon receiving the *logical object ID* of the video of interest from PREDATOR, the *Quality Manager* of QuaSAQ annotates a series of plans for QoS-guaranteed delivery and chooses one to execute. It communicates with either QuaSAQ modules in remote sites or local Shore component (depending on the plan it chooses) to retrieve the video of interest. Note the sender of the video data is not necessarily the site at which the query was received and processed.

4.2 QuaSAQ Components

Most of the QuaSAQ components are developed by modifying and augmenting relevant modules in VDBMS (e.g. client program, SQL parser, query optimizer). As described in Section 3, the realization of QuaSAQ also depends on the availability of some other software modules that are not found in VDBMS.

Replicas for all videos in the database are generated using a commercial video transcoding/encoding software VideoMach[5]. The choice of quality parameters is determined in a way that the bitrate of the resulting video replicas fit the bandwidth of typical network connections such as T1, DSL, and modems (Table 2). To obtain an online video transcoder, we modify the source code of the popular Linux video processing tool named *transcode*[6] and integrate it into the *Transport API* of QuaSAQ. The major part of the *Transport API* is developed on the basis of a open-source media streaming program[7]. It decodes the layering information of MPEG stream files and leverages the synchronization functionality of the Real Time Protocol (RTP) by encapsulating video stream load into RTP packets. We also implemented various frame dropping strategies for MPEG1 videos as part of the Transport API.

The existence of QoS resource management (Composite QoS APIs) that is capable of reservation, monitoring and adaptation is the most important assumption in the design of QuaSAQ. Resource QoS support can be achieved on two levels: on the kernel of a QoS-enabled Operating System [28] and on the application level (middleware) [18]. Although the OS solution gives more efficient and accurate QoS support, the middleware is easier to develop and deploy. To take advantage of such convenience, we build the Composite QoS APIs using a QoS-aware middleware named GARA [7] as substrate. GARA features a series of simple and unified APIs and contains separate resource managers for individual resources (e.g. CPU, network bandwidth and storage bandwidth). For example, the CPU manager in GARA is based on the application-level CPU scheduler DSRT [4] developed in the context of the QualMan project [17]. The management of network bandwidth is more complicated: it requires not only the deployment of resource management modules on the end-point systems but the participation of network routers as well. QoS-aware network protocols are generally the solution to this problem. In GARA, the *Diff-Srv* mechanism provided by the Internet Protocol (IP) is used.

5. EXPERIMENTAL RESULTS

We evaluated the performance of QuaSAQ in comparison with the original VDBMS system. The experiments are focused on the QoS improvement/degradation in video delivery as well as system throughput. An important metric in measuring QoS of networked video streaming tasks is the *inter-frame delay*, which is defined as the interval between the *processing time* of two consecutive frames in a video stream [4, 28]. Ideally, the inter-frame delay should be the reciprocal of the frame rate of the video. Deviations of the inter-frame delay from its theoretical value are generally compensated for by client-side memory buffers. The larger the deviation, the larger the buffer needed. We also consider the packet loss rate as a metric in measuring QoS. For the system throughput, we simply use the number of concurrent streaming sessions and the reject rate of queries.

Experimental setup. The experiments are performed on a small distributed system containing three servers and a number of client machines. The servers are all Intel machines (one Pentium 4 2.4GHz CPU and 1GB memory) running Solaris 2.6. The servers are located at three different 100Mbps Ethernets, two of which are in the domain of cs.purdue.edu and one in ecn.purdue.edu. Each server has a total streaming bandwidth of 3200KBps. The clients are deployed on machines with various hardware configurations generally 2-3 hops away from the servers. Due to lack of router support of the *DiffSrv* mechanism, only *admission control* is performed in network management. A reasonable assumption here is that the bottlenecking link is always the outband link of the severs [22] and those links are dedicated for our experiments. Instead of user inputs from a GUI-based client program [1], the queries for the experiments are from a traffic generator. Every synthesized query consists of a destination (client) ID, an OID of a video to be streamed, and the QoS requirements (We bypass the QoP stage since the main purpose is to evaluate the Quality Manager and QoS APIs). Our experimental video database contains 15 videos in MPEG1 format with playback time ranging from 30 seconds to 18 minutes. For each video, three to four QoS-specific replicas are generated and fully replicated on three servers so that each server has all copies. In our experiments, we only utilized a thin client program that passively receives video packets from the servers and collects statistics about the received data.

5.1 Improvement of QoS by QuaSAQ

Figure 7 shows the inter-frame delay of a representative streaming session for a video with frame rate of 23.97 fps. The data is collected on the server side, e.g. the *processing time* of a video frame is when it is first handled. The server-side inter-frame delay is a good indication of the timeliness of the pre-transmission video processing, or *server activities* (Section 3.4.1). Only end-point system resources should be considered here (no need to transfer videos between servers due to full replication). The left two graphs of Figure 7 represent the result of the original VDBMS while the right two graphs show those with QuaSAQ. We compare the performance of both systems by their response to various contention levels. On the first row, streaming is done without competition from other programs (low contention) while the number of concurrent video streams are high (high contention) for experiments on the second row.

Table 3: Statistics of Inter-frame and Inter-GOP delays shown in Figure 7. Unit for all data is millisecond, S.D. = Standard Deviation.

Tested System & Contention	*Inter-frame*		*Inter-GOP*	
	Mean	*S. D.*	*Mean*	*S. D.*
VDBMS, Low	42.07	34.12	622.82	64.51
VDBMS, High	48.84	164.99	722.83	246.85
QuaSAQ, Low	42.16	30.89	624.84	10.13
QuaSAQ, High	42.25	30.29	626.18	8.68

Under low contention, both systems (Fig 7a and 7b) demon-

[5]Release 2.6.3, http://www.gromada.com

[6]Release 0.6.4, http://www.theorie.physik.uni-goettingen.de/ ostreich/transcode/

[7]www.live.com

Table 2: *QoS of typical video replicas in QuaSAQ prototype.*

Target Standard	*Bit Rate (KB/s)*	*Color (bit)*	*Resolution (pixels)*	*Frame Rate (fps)*	*Audio Sample Rate (Hz)*
VCD (NTSC)	113	24	352×240	29.97	44100
DSL	81	24	320×200	29.97	32000
ISDN-2x	24	24	176×112	29.97	32000

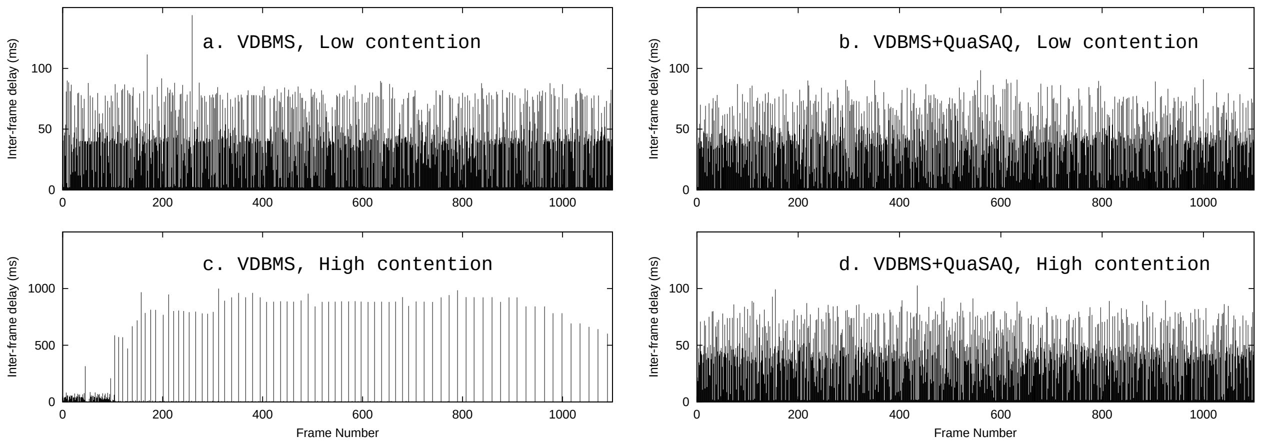


Figure 7: Inter-frame delays on the server side under different system contentions.

strated timely processing of almost all the frames, as shown by their relatively low variance of inter-frame delay (Table 3). Note that some variance are inevitable in dealing with Variable Bitrate (VBR) media streams such as MPEG video because the frames are of different size and coding scheme. In our example with MPEG streams, the time needed to handle I-frames is always longer than the following B and P frames in each *Group of Pictures* (GOP). This is indicated by the periodically appearing high lines in Figure 7a and 7b. Such intrinsic variance can be smoothed out if we collect data on the GOP level (Table 3). The compatible performance of QuaSAQ and VDBMS shows that real-time jobs such as video streaming, although scheduled as time-sharing processes, can achieve good timeliness when there are only minimal competitions. Under such circumstances, deployment of QoS mechanisms has no advantage.

VDBMS shows very different behavior from the QuaSAQ-enhanced database under high contention. Its variance of inter-frame delays (Fig 7c) are huge as compared to those of QuaSAQ (Fig 7d). Note the scale of the vertical axis in Figure 7c is one magnitude higher than that of all three other diagrams. The reason for such high variance is poor guarantee of CPU cycles for the streaming jobs. The job waits for its turn of CPU utilization at most of the time. Upon getting control over CPU, it will try to process all the frames that are overdue within the quantum assigned by the OS (10ms in Solaris). Therefore, VDBMS processes video in a chunk-by-chunk pattern instead of frame-by-frame, as we can see from the deep valleys in Fig 7c. Besides high variance, the average inter-frame delay is also big for VDBMS under high contention (Table 3). Note the theoretical inter-frame delay for the sample video is $1/23.97 = 41.72ms$. Increased average inter-frame delay is an obvious sign of QoS degradation: it shows the lack of adequate resource (CPU) to handle all jobs concurrently so that the total streaming time is longer than expected. To the human user on the client side, this could result in jitter/interruption during video playback. On the contrary, QuaSAQ achieves similar performance when system contention level changes. With the help of QoS APIs, the CPU needs in QuaSAQ are highly guaranteed, resulting in timely processing of video frames on the end-point machines.

Table 4: Statistics of Inter-frame delay and loss rate on the client side. SD = Standard Deviation.

Tested Systems	*Inter-frame Delay*		*Loss Rate*
	Accuracy	*SD/Mean*	*(%)*
VDBMS	1.22	3.29	29.8
VDBMS+QuaSAQ	1.002	0.633	0.012
VDBMS+QoS API	1.001	0.265	0.001

Client side QoS. Inter-frame delays are also collected and analyzed on the client-side (Table 4). Unlike the server-side data, client-side Inter-frame delay is a reflection of resource availability in both end-point systems and the network. The latter can also be demonstrated by loss rate of the UDP-based RTP packets for video streaming in the tested systems. The data in Table 4 are collected in two client machines from which details of 6 to 8 sample streams are recorded. For a streaming session, the *Accuracy* is defined as the ratio of the average to theoretical value of inter-frame delays (see Table 3) while *SD/Mean* is the ratio of standard deviation to the average. Data in each cell is the average of the 6 to 8 sample streams. As compared to VDBMS, the

QuaSAQ-enhanced system achieves much better end-to-end QoS, as shown by its high accuracy and low deviation. The packet loss is also minimal, as a result of guaranteed network bandwidth. The third system, denoted as *VDBMS+QoS API* (details in Section 5.2), has very similar performance to QuaSAQ.

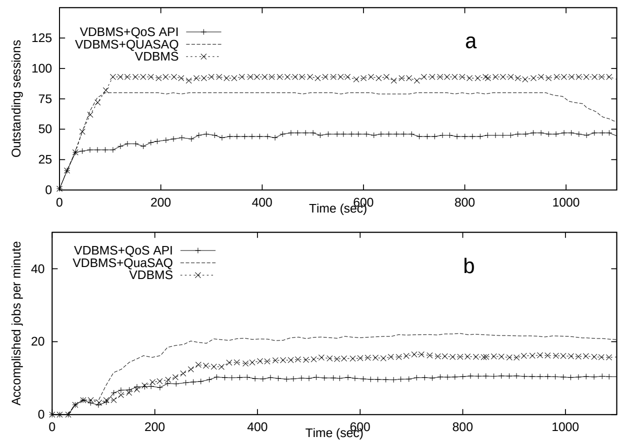


Figure 8: Throughput of different video database systems. a. Concurrent sessions supported; b. Number of accomplished sessions per minute.

5.2 System throughput

We compare the throughput of three different experimental systems: VDBMS, VDBMS with QuaSAQ, and VDBMS equipped with Composite QoS APIs (Fig 8). In the last system, only resource reservation and admission control are performed in addition to basic VDBMS functionalities. The same set of queries are fed into the three tested systems. Queries are generated such that the access rate to each individual video is the same and each QoS parameter (QuaSAQ only) is uniformly distributed in its valid range. The inter-arrival time for queries is exponentially distributed with an average of 1 second. The original VDBMS obviously keeps the largest number of concurrent streaming sessions (Fig 8a). However, the seemingly high throughput of VDBMS is just a result of lack of QoS control: all video jobs were admitted and it took much longer time to finish each job (Table 3). To avoid an unfair comparison between VDBMS and QuaSAQ, a VDBMS enhanced with QoS APIs is introduced. The streaming sessions in both systems are of high quality (Section 5.1). According to Figure 8a, throughput for all three systems stabilize after a short initial stage. QuaSAQ beats the "VDBMS + QoS API" system by about 75% on the stable stage in system throughput. This clearly shows the advantages of QoS-specific replication and Quality Manager that are unique in QuaSAQ. The superiority of QuaSAQ is also demonstrated in Figure 8b where we interpret throughput as the number of succeeded sessions per unit time. The throughput on stable stage of QuaSAQ is shown to be about 100% and 37% higher than that of VDBMS+QoS API and VDBMS, respectively.

We also evaluate our resource-based cost model (Fig 9). We compare the throughput of two QuaSAQ systems using different cost models: one with the Lowest Resource Bucket

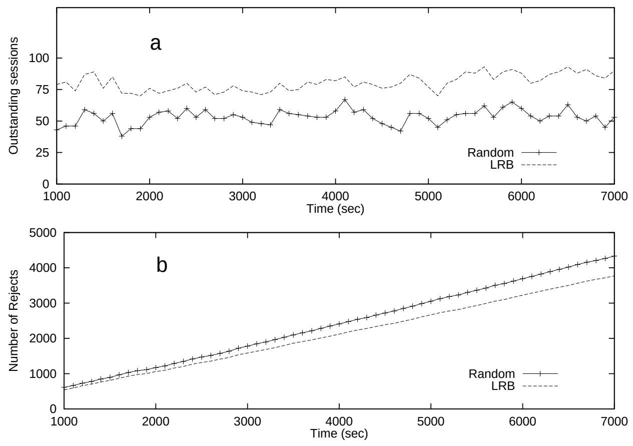


Figure 9: Throughput of QuaSAQ systems with different cost models. a. Concurrent sessions supported; b. Number of rejected queries

model and one with a simple randomized algorithm. The latter randomly selects one execution plan from the search space. The randomized approach is a frequently-used query optimization strategy with fair performance. Without performance being significantly better than that of the randomized approach, a newly-proposed cost model can hardly be regarded successful. The queries are generated in the same way as those in the previous experiment (Fig 8). It is easy to see that the resource-based cost model achieves much better throughput (Fig 9a). The number of sessions supported is 27% to 89% higher than that of the system with the randomized method. The high system throughput caused by the proposed cost model is also consistent with its low reject rate shown in Figure 9b.

To thoroughly test the validity of the resource-based cost model, we measure the throughput under various environments (Fig 10). In figure 10a, we decreased the total streaming bandwidth of three servers to 1MBps. This makes the network bandwidth a bottleneck resource at all times. In figure 10b, the access pattern to media content becomes extremely skewed. Traffic are generated using an access model called 90/10 where 90% of the requests ask for only 10% of the media objects. The inter-arrival time of queries in figure 10c is 5 seconds instead of 0.5 seconds in previous experiments. In figure 10d, we produce queries of more specific QoS requirements by decreasing the range of QoP to QoS mapping. As a result, the number of candidate plans for each query is lowered. One common features of all these treatments is that the flexibility in plan generation is reduced so that the advantages of our proposed model could be masked. However, LRB still shows some improvement of throughput over the randomized model. Not in one single point is the number of sessions supported in LRB smaller than that of the randomized model.

5.2.0.2 Overhead of QuaSAQ..

QuaSAQ is a light-weight extension to VDBMS. The throughput data in Figure 8 already show that the overhead for running QuaSAQ does not affect performance. The major cost of QuaSAQ comes from the CPU cycles used for mainte-

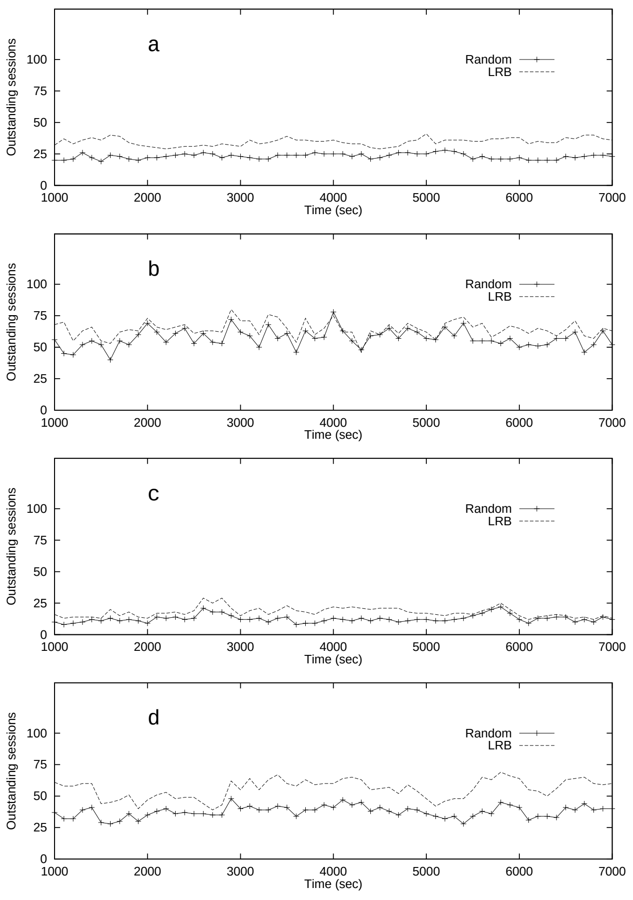


Figure 10: Throughput of QuaSAQ systems under different environmental situations.

nance of the underlying QoS resource management modules. The DSRT scheduler reports an overhead of $0.4 - 0.8ms$ for every $10ms$ [5]. This number is only $0.16ms$ in the machines we used for experiments (1.6% overhead). The CPU use for QoS-aware query processing is on the order of milliseconds. This reflects the cost for the construction and evaluation of generally 30-40 plans for each query. Being a per-query cost, it is overshadowed by the cost of DSRT. No network resource is consumed in the tested system where only admission control is performed by the Composite QoS API.

6. RELATED WORK

Numerous research projects have been dedicated to the theory and realization of QoS control on the lower (system, network) levels [17, 7, 9, 28]. On the contrary, research on user-level QoS has attracted less attention and left us many open issues. In our QoS-aware distributed multimedia database, we assume the low-level QoS functionalities are given in the form of a *Composite API*. The Multimedia Support Infrastructure (MSI)[8] and Indiana Telemedicine Incubator (ITI)[9] projects currently underway at Purdue University are examples of attempts at providing user-level quality of service over collections of distributed multimedia repositories.

The software releases of QualMan [17] and GARA [7] projects are the foundations upon which we build our low level QoS APIs. In QualMan, access to shared resource is controlled by a resource broker in a client/server model. For any brokerage request, the client and server negotiate towards a QoS contract and the server broker performs resource admission control. Then QoS contract is sent to relevant *resource scheduler* for fulfillment. Unlike the strictly reservation based and a focus on end-system QoS, GARA gives more weight to resource adaptation and provides solutions for network QoS by using the *DiffSrv* mechanisms over IP.

Our goals are close in spirit to those of similar projects at the University of Illinois [14, 13]. A key difference lies in our database-centric approach versus the compiler and distributed objects approach. In our approach, QoS requirements are integrated into database queries. Based on a pool of replicated media data with different quality characteristics, the enhanced query processor (QuaSAQ) dynamically generates multiple alternative plans that are evaluated at run time. Our solution therefore trades greater flexibility in finding good execution plans with increased run-time cost in comparison to the approach of compiling a plan picked by an application designer. We also show the validity of a novel cost model based on each plan's resource use. Using this model, QuaSAQ chooses plans that maximize system resource utilization and also meet the QoS requirements specified by the users.

The following pieces of work are directly related to our QoS-aware multimedia DBMS. In [8], a QoS management framework for distributed multimedia applications is proposed with the focus of dynamic negotiation and translation of user-level QoS by QoS profiling. The same group also presents a generic framework for processing queries with QoS constraints in the context of conventional DBMS [29]. They emphasize the need to evaluate queries based on a *QoS-based cost model* that takes system performance into account. However, the paper lacks technical details on how to develop these cost models. Nor does it provide any experimental data for the validation of its methodology. A conceptual model for QoS management in multimedia database systems is introduced in [25]. In this paper, QoS is viewed as the distance between an actual presentation and the ideal presentation (with perfect quality) of the same media content. The metric space where the distances are defined consists of n dimensions, each of which represents a QoS parameter. *Utility functions* are used to map QoS into a satisfaction value, either on a single dimension or all QoS as a whole. The paper also proposes a language for QoS specification. Although the architecture of a prototype utilizing their QoS model is illustrated, further details on implementation and evaluation of the system are not discussed. Our work on QoS differs from [25] in two aspects: we focus on the change of query processing mechanisms in multimedia DBMS while they are more inclined to QoS semantics on a general multimedia system; we invest much effort in experimental issues while they introduce only a theoretical framework.

The design and realization of QuaSAQ is motivated by the high-level concepts sketched in a previous work [2]. The main contribution of [2] is to specify QoS in video database queries by a query language based on constraint logic pro-

[8]http://www.cs.purdue.edu/msi
[9]http://www.cs.purdue.edu/iti

gramming. They propose the *content* and *view* specifications of queries. The former addresses correctness while the latter captures the quality of the query results. The paper also discusses the modifications to current multimedia DBMS design in order to accommodate the new feature of QoS management. Similar to [25], the paper concentrates on building a logical framework rather than the design and implementation of a real-world system.

The idea of *Dynamic Query Optimization* [12, 20] is analogous to QoS renegotiation in QuaSAQ. Both involve finding a better plan in response to change of system status. However, the renegotiation process in QuaSAQ is more likely to be triggered by change of user intention than resource availability. Furthermore, traditional time-based cost models are used in the aforementioned studies while we build our resource-based model in QuaSAQ. Other related efforts include: [30] studies cost estimation of queries under dynamic system contentions; [24] discusses QoS control for general queries in real-time databases; various dynamic replication strategies in a video-on-demand environment are presented and evaluated in [15].

7. CONCLUSIONS AND FUTURE WORK

We have presented an overview of our approach to enabling end-to-end QoS for distributed multimedia databases. We discussed various issues pertaining to design and implementation of a QoS-aware query processor (QuaSAQ) with the focus of novel query evaluation and optimization strategies. As a part of the query processing scheme of QuaSAQ, we presented a novel cost model that evaluates query plans by their resource consumption. QuaSAQ was implemented and evaluated on the context of the VDBMS project. Experimental data demonstrated the advantages of QuaSAQ in two aspects: highly improved QoS guarantee and system throughput.

We are currently in the process of implementing a more complete version of QuaSAQ as part of our ongoing projects. This includes efforts to add more resource managers in the Composite QoS API, security mechanisms, and more refined plan generator and cost models. The QuaSAQ idea also needs to be validated on distributed systems with scales larger than the one we deployed the prototype on. On the theoretical part, we believe the refinement and analysis of the resource-based cost model is a topic worthy of further research. The extension of the applicability of our cost model from video databases to more general multimedia and database environments is also promising.

8. REFERENCES

[1] W. Aref, A. C. Catlin, A. Elmagarmid, J. Fan, J. Guo, M. Hammad, I. F. Ilyas, M.S. Marzouk, S. Prabhakar, A. Rezgui, E. Terzi, Y. Tu, A. Vakali, and X.Q. Zhu. A Distributed Database Server for Continuous Media. In *Proceedings of the 18th ICDE Conference*, pages 490–491, February 2002.

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

[3] M. Carey, D. DeWitt, M. Franklin, N. Hall, M. McAuliffe, J. Naughton, D. Schuh, M. Solomon, C. Tan, O. Tsatalos, S. White, and M. Zwilling. Shoring Up Persistent Applications. In *Proceedings of ACM SIGMOD*, pages 383–394, 1994.

[4] H-H. Chu and K. Nahrstedt. A Soft Real Time Scheduling Server in UNIX Operating System. In *Proceedings of IDMS*, pages 153–162, 1997.

[5] Hao-Hua Chu. *CPU Service Classes: A Soft Real Time Framework for Multimedia Applications*. PhD thesis, University of Illinois at Urbana-Champaign, 1999.

[6] Beverly Chua. Impact of Spatial-temporal resolution on video quality. Technical Report TR-03/34, CSRIO Mathematical and Information Sciences, 2003.

[7] I. Foster, A. Roy, and V. Sander. A Quality of Service Architecture that Combines Resources Reservation and Application Adaptation. In *Proceedings of IWQOS*, pages 181–188, June 2000.

[8] A. HAfiD and G. Bochmann. An Approach to Quality of Service Management in Distributed Multimedia Application: Design and Implementation. *Multimedia Tools and Applications*, 9(2):167–191, 1999.

[9] J. Huang, R. Jha, W. Heimerdinger, M. Muhammad, S. Lauzac, and B. Kannikeswaran. RT-ARM: A Real-Time Adaptive Resource Management System for Distributed Mission-Critical Applications. In *Proceedings of the IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, December 1997.

[10] Ramesh Jain and Arun Hampapur. Metadata in Video Databases. *SIGMOD Record*, 23(4):27–33, 1994.

[11] H. Jiang and A. K. Elmagarmid. Spatial and temporal content-based access to hyper video databases. *The VLDB Journal*, 7(4):226–238, 1998.

[12] N. Kabra and D. DeWitt. Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans. In *Proceedings of ACM SIGMOD*, pages 106–117, 1998.

[13] B. Li and K. Nahrstedt. A Control-Based Middleware Framework for Quality of Service Adaptations. *IEEE Journal of Selected Areas in Communications, Special Issue on Service Enabling Platforms*, 1999.

[14] B. Li, D. Xu, and K. Nahrstedt. An Integrated Runtime QoS-Aware Middleware Framework for Distributed Multimedia Applications. *ACM Multimedia Systems Journal, Special Issue on Multimedia Middleware*, 8(5):420–430, 2002.

[15] P. W. K. Lie, J. C. S. Lui, and L. Golubchik. Threshold-Based Dynamic Replication in Large-Scale Video-on-Demand Systems. *Multimedia Tools and Applications*, 11(1):35–62, 2000.

[16] G. Menges. *Economic Decision Making: Basic Concepts and Models*, chapter 2, pages 21–48. Longman, 1973.

[17] K. Nahrstedt, H. Chu, and S. Narayan. QoS-Aware Resource Management for Distributed Multimedia Applications. *Journal on High-Speed Networking, Special Issue on Multimedia Networking*, 8(3–4):227–255, 1998.

[18] K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li. QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments. *IEEE Communications Magazine*, 2001.

[19] Klara Nahrstedt and Ralf Steinmetz. Resource

Management in Networked Multimedia Systems. *IEEE Computer*, 28(5):52–63, 1995.

[20] Kenneth W. Ng, Zhenghao Wang, Richard R. Muntz, and Silvia Nittel. Dynamic Query Re-Optimization. In *Proceedings of SSDBM*, pages 264–273, 1999.

[21] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*, chapter 9, pages 228–273. Prentice Hall, 1999.

[22] V. N. Padmanabhan and K. Sripanidkulchai. The Case for Cooperative Networking. In *Proceedings of he First International Workshop on Peer-to-Peer Systems (IPTPS)*, March 2002.

[23] P. Seshadri. Enhanced Abstract Data Types in Object-Relational Databases. *The VLDB Journal*, 7(3):130–140, 1998.

[24] J. Stankovic, S. Son, and J. Liebeherr. Beehive: Global multimedia database support for dependable, real-time applications. In A. Bestavros and V. Fay-Wolfe, editors, *Real-Time Database and Information Systems: Research Advances*, chapter 22, pages 409–422. Kluwer Academic Publishers, 1997.

[25] J. Walpole, C. Krasic, L. Liu, D. Maier, C. Pu, D. McNamee, and D. Steere. Quality of Service Semantics for Multimedia Database Systems. In *Proceedings of Data Semantics 8: Semantic Issues in Multimedia Systems IFIP TC-2 Working Conference*, volume 138, 1998.

[26] L. Wolf, C. Gridwodz, and R. Steinmetz. Multimedia Communication. *Proceedings of the IEEE*, 85(12):1915–1933, December 1997.

[27] T. Yamazaki and J. Matsuda. On QoS Mapping in Adaptive QoS Management for Distributed Multimedia Applications. In *Proceedings of ITC-CSCC'99*, pages 1342–1345, 1999.

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

[29] H. Ye, B. Kerhervé, and G. v. Bochmann. Quality of Service Aware Distributed Query Processing. In *Proceedings of DEXA Workshop on Query Processing in Multimedia Information Systems(QPMIDS)*, September 1999.

[30] Q. Zhu, S. Motheramgari, and Y Sun. Cost Estimation for Queries Experiencing Multiple Contention States in Dynamic Multidatabase Environments. *Knowledge and Information Systems*, 5(1):26–49, 2003.