QuaSAQ: An Approach to Enabling End-to-End QoS for Multimedia Databases

Yi-Cheng Tu, Sunil Prabhakar, Ahmed Elmagarmid, Radu Sion

Purdue University, West Lafayette IN 47907, USA

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. Our proposed architecture to enable end-to-end QoS control, the QoS-Aware Query Processor (QuaSAQ), satisfies user specified quality requirements. The users need not be aware of detailed low-level QoS parameters, but rather specifies high-level, qualitative attributes. In addition to an overview of key research issues in the design of QoS-aware databases, this paper presents our proposed solutions, and system implementation details. An important issue relates to the enumeration and evaluation of alternative plans for servicing QoS-enhanced queries. This step follows the conventional query execution which results in the identification of objects of interest to the user. We propose a novel *cost model* for media delivery that explicitly takes the resource utilization of the plan and the current system contention level into account. Experiments run on the QuaSAQ prototype show significantly improved QoS and system throughput.

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 delivery 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 DBMS design, one has to be concerned about not only the *correctness* but also the *quality* of the query results. The set of quality parameters that describes the temporal/spatial constraints of media-related applications is called *Quality of Service* (QoS) [1]. 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.

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 [2], little effort has been devoted to the aspect of the integration of QoS support. 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 [3]. Simply deploying a multimedia DBMS on top of a QoS-provisioning system will not provide end-to-end QoS. Moreover, such a solution is unable to exploit the application level flexibility such as the user's acceptable range of quality. 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 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"). 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 our QoS-aware multimedia DBMS. 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, this is the first prototype system that achieves end-to-end QoS for multimedia databases. We present experimental results from our prototype that establish the feasibility and advantages of such a system. Our implementation builds upon the VDBMS prototype multimedia database system developed by our group at Purdue University [4]. Among other enhancements, QuaSAQ extends VDBMS to build a distributed QoS-aware multimedia DBMS with multiple copies of storage/streaming manager.

To address the structure of a QoS-provisioning networked multimedia system, four levels of QoS have been proposed: user QoS, application QoS, system QoS, and network QoS [1, 5]. We consider a series of QoS parameters in our research as shown in Table 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. Most current research on QoS fail to address the optimization of system performance. In this paper, 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 management 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

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*

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 runtime QoS adaptation options (e.g. frame dropping), 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 module that enables 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 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 current system status; 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 DBMS 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. In order to extend the D-DBMS approach to address end-to-end QoS, 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. 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 the storage space.

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. 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.
3. A model for the *search space* of possible execution plans. The search space 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*. 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* [6]. 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.
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.
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* [1]. 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 addresses all 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] Sometimes *response time* is also used, as in distributed INGRES.

2 and 5, we concentrate on the implementation and evaluation of known approaches within the context of multimedia databases. Item1 will be covered in a follow-up paper.

3 Quality-of-Service Aware Query Processor (QuaSAQ)

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.

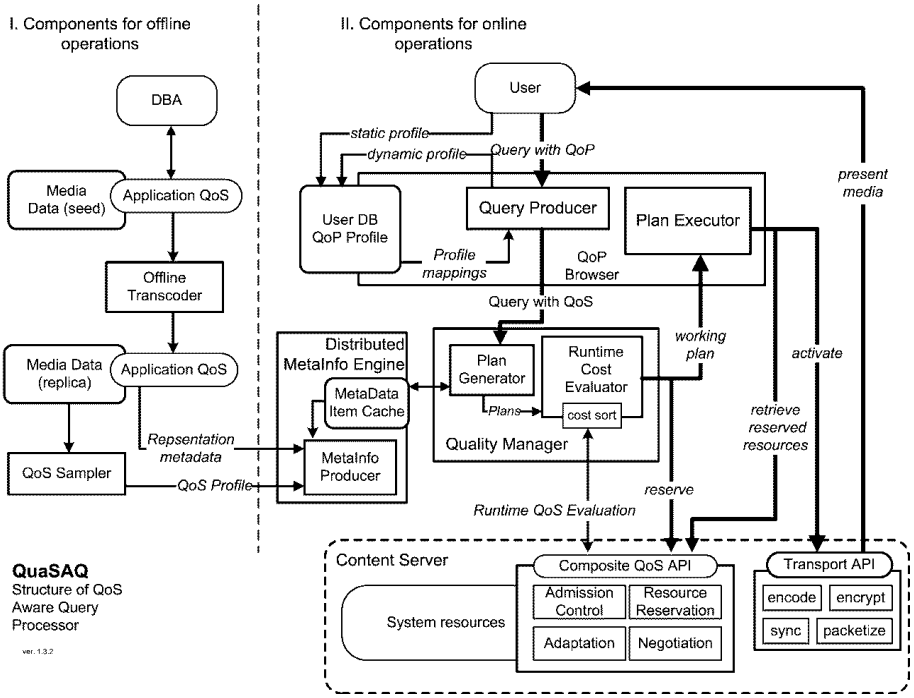

Fig. 1. QuaSAQ architecture

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 those, 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* as metadata. Please refer to [7] for more details of replication and QoS mapping in QuaSAQ.

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. It basically performs actual presentation, synchronization as well as runtime maintenance of underlying QoS parameters.

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, and audio quality. 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 $320\times240 - 352\times288$ pixels. The application QoS parameters are quantitative and we achieve some flexibility by allowing one QoP mapped to a range of QoS values. QoS requirements are allowed to be modified during media playback and a renegotiation is expected. 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.

One important weakness of these qualitative formulations of QoP is their lack of flexibility (i.e. failure to capture 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.

3.3 Distributed Metadata Engine

In a multimedia DBMS, operations such as content-based searching depend heavily, if not exclusively, on the metadata of the media objects [2]. As mentioned in Section 2, video objects are stored in several locations, each copy with different representation characteristics. This requires more items in the metadata collection. Specifically, we require at least the following types of metadata for a QoS-aware DBMS:

– *Content Metadata:* describe the content of objects to enable multimedia query, search, and retrieval. In our system, a number of visual and semantic descriptors such as shot detection, frame extraction, segmentation, and camera motion are extracted.
– *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. It records the OIDs of objects and the mapping between media content and media file.
– *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.

We distribute the metadata in various locations 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:

Plan Generator. The *Plan Generator* is in charge of generating plans that enable the execution of the query from the Query Producer. 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). A given logical object may be replicated at multiple sites and further with different formats. For example, a given video may be stored in different resolutions and color depth at two different sites. The plan generator determines which of the alternatives can be used to satisfy the request and also the necessary steps needed to present it to the user.

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. 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 $O(n!d^n)$. Fortunately, there are also some other system-specific rules that further reduce the number of alternative plans. One salient rule is related 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 $O(d^n)$.

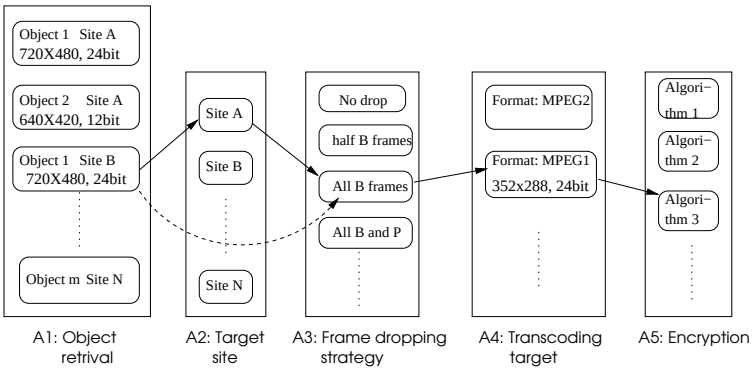

Fig. 2. Illustrative plan generation in QuaSAQ

Runtime QoS Evaluation and Plan Drop. The Plan Generator described above does not check generated plans for any QoS constraints. 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 inputs 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. Therefore, 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 requirements (in the form of a *resource vector*) and feeds it to the next component down the processing pipe-line.

Illustrative examples of plans. The path in solid lines shown in Figure 2 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. transcode to MPEG1 format with certain target QoS; 4. drop all the B frames during delivery; 5. encrypt the media data using algorithm 1. The dotted line corresponds to a simpler plan: retrieve the same object and transcode with the same target QoS, no frame dropping or encryption is needed. An even simpler plan would be a single node in set $A1$, meaning the object is sent without further processing.

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 first plan in this order that satisfies the QoS requirements is 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 QuaSAQ, the total time for executing any query plans is exactly the same since the streaming time for a media object is fixed. As a result, processing time is no longer a valid metric for cost estimation of the QoS-aware query plans.

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 bandwidth. 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. Furthermore, the cost model id also 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:

$$E = \frac{G}{C(\mathbf{r})}$$

where C is the cost function, $\mathbf{r}$ 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 [8]. A detailed discussion of the configurable cost model mentioned above is beyond the scope of this paper. Instead, we present a simple cost model that aims to maximize system throughput.

Lowest Resource Bucket (LRB) model. Suppose there are n types of resources to be considered in QuaSAQ, we denote the total amount of resource i as R_i. In our algorithm, we build a virtual *resource bucket* for each individual

resource. All R_i values are standardized into the height of the buckets. Therefore, the height of all buckets is 1 (or 100%). The buckets are filled when the 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 3d has height 42, which means 42% of R_2 is currently in use. 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. In Figure 3, 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* of the LRB model can be expressed as

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

where U_i is the current usage of resource i. The input is the resource vector of the plan being evaluated.

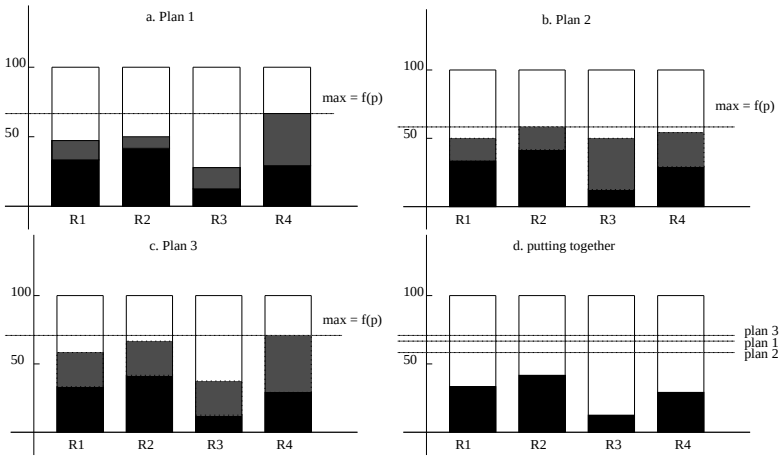

Fig. 3. Cost evaluation by the Lowest Resource Bucket model

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

The *Composite QoS API* hides implementation and access details of underlying APIs (i.e. system and network) and offers control to upper layers (e.g. *Plan Generator*) at the same time. The major functionality provided by 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; 2.*Resource reservation,* an important strategy toward QoS control by guaranteeing resources needed during the lifetime of media delivery jobs; 3.*Renegotiation* that are mainly performed under two scenarios mentioned in Section 3.2.

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 Composite QoS 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 [4]. The QuaSAQ development is done using C++ under the Solaris 2.6 environment. Figure 4 shows the architecture of VDBMS enhanced with the QuaSAQ prototype.

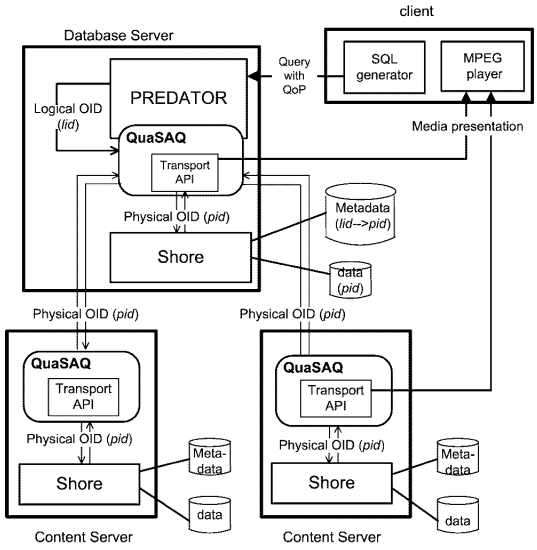

Fig. 4. System architecture for QoS-aware multimedia DBMS prototype

QuaSAQ and VDBMS. Developed from the object-relational database engine PREDATOR[9] with Shore[10] as the underlying storage manager (SM), VDBMS is a multimedia DBMS that supports full-featured video operations (e.g. content-based searching, streaming) and complex queries. Most of the VDBMS development was done by adding features to PREDATOR. We extended the current release of VDBMS, which runs only on a single node, to a distributed version by realizing communication and data transferring functionalities among different sites. As shown in Figure 4, 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 [3]. In VDBMS, the query processor returns an object ID (OID), by which Shore retrieves the video from disk. With QuaSAQ, these OIDs refer to the video content (represented by logical OID) rather than the entity in storage (physical OID) since multiple copies of the same video exist. In QuaSAQ, the mapping between logical OIDs and physical OIDs are stored as part of the metadata (Section 3.3). Upon receiving the logical OID 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 initiate the video transmission. Note the sender of the video data is not necessarily the site at which the query was received and processed.

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). Replicas for all videos in the database are generated using a commercial video transcoding/encoding software VideoMach[2]. 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 [7]. To obtain an online video transcoder, we modified the source code of the popular Linux video processing tool named *transcode*[3] and integrated it into the *Transport API* of our QuaSAQ prototype. The major part of the *Transport API* is developed on the basis of a open-source media streaming program[4]. It decodes the layering information of MPEG stream files and leverages the synchronization functionality of the Real Time Protocol (RTP). We also implement various frame dropping strategies for MPEG1 videos as part of the Transport API. We build the Composite QoS APIs using a QoS-aware middleware named GARA [11] as substrate. GARA contains separate managers for individual resources (e.g. CPU, network bandwidth and storage bandwidth). The CPU manager in GARA is based on the application-level CPU scheduler DSRT [12] developed in the context of the QualMan project [13]. QoS-aware network protocols are generally the solution to network resource management,

[2] Release 2.6.3, http://www.gromada.com
[3] Release 0.6.4, http://www.theorie.physik.uni-goettingen.de/∼ostreich/transcode/
[4] http://www.live.com

which requires participation of both end-systems and routers. In GARA, the *DiffSrv* 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 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 [12, 14]. Ideally, the inter-frame delay should be the reciprocal of the frame rate of the video. 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 in the domain of purdue.edu. Each server has a total streaming bandwidth of 3200KBps. The clients are deployed on machines that are 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 and those links are dedicated for our experiments. Instead of user inputs from a GUI-based client program [4], the queries for the experiments are from a traffic generator. Our experimental video database contains 15 videos in MPEG-1 format with playback time ranging from 30 seconds to 18 minutes. For each video, three to four copies with different quality are generated and fully replicated on three servers so that each server has all copies.

5.1 Improvement of QoS by QuaSAQ

Figure 5 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* is when the video frame is first handled. Only end-point system resources should be considered in analyzing server-side results. The left two graphs of Figure 5 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.

Under low contention, both systems (Fig 5a and 5b) demonstrated timely processing of almost all the frames, as shown by their relatively low variance of inter-frame delay (Table 2). Note that some variance are inevitable in dealing with Variable Bitrate (VBR) media streams such as MPEG video because the

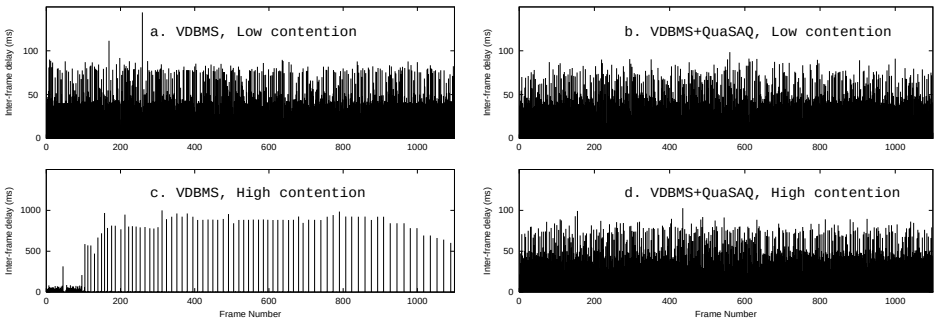

Fig. 5. Inter-frame delays on the server side under different system contentions

frames are of different sizes and coding schemes (e.g. I, B, P frames in a Group of Pictures (GOP) in MPEG). Such intrinsic variance can be smoothed out if we collect data on the GOP level (Table 2).

Table 2. Statistics of Inter-frame and Inter-GOP delays shown in Figure 5. Unit for all data is millisecond, S.D. = Standard Deviation.

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

VDBMS is unable to maintain QoS under high contention. Its variance of inter-frame delays (Fig 5c) are huge as compared to those of QuaSAQ (Fig 5d). Note the scale of the vertical axis in Figure 5c is one magnitude higher than those of 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). Besides high variance, the average inter-frame delay is also large for VDBMS under high contention (Table 2). Note the theoretical inter-frame delay for the sample video is $1/23.97 = 41.72ms$. 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. Data collected on the client side show similar results [7].

5.2 System throughput

We compare the throughput of QuaSAQ and the original VDBMS (Fig 6). The same set of queries are fed into the 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 6a). 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 2). To avoid an unfair comparison between VDBMS and QuaSAQ, a VDBMS enhanced with QoS APIs is introduced. The streaming sessions in this system are of the same (high) quality as those in QuaSAQ (data not shown). According to Figure 6a, 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 6b where we interpret throughput as the number of succeeded sessions per unit time.

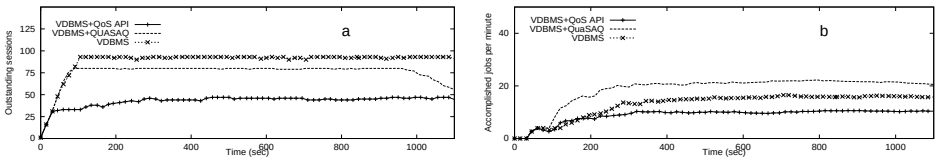

Fig. 6. Throughput of different video database systems

We also evaluate our resource-based cost model (Fig 7). We compare the throughput of two QuaSAQ systems using different cost models: one with LRB 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 6). It is easy to see that the resource-based cost model achieves much better throughput (Fig 7a). 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 7b.

Overhead of QuaSAQ. QuaSAQ is a light-weight extension to VDBMS. The throughput data in Section 5.2 already show that the overhead for running QuaSAQ does not affect performance. The major cost of QuaSAQ comes from the CPU cycles used for maintenance of the underlying QoS resource management modules. The DSRT scheduler reports an overhead of $0.4-0.8ms$ for every $10ms$ [12]. This number is only $0.16ms$ in the machines we used for experiments (1.6% overhead). The CPU use for processing each query (a few milliseconds) in QuaSAQ is negligible.

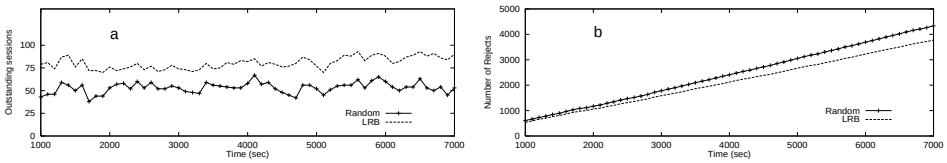

Fig. 7. Throughput of QuaSAQ systems with different cost models

6 Related Work

Numerous research projects have been dedicated to the theory and realization of QoS control on the lower (system, network) levels [13, 11, 14]. The software releases of QualMan [13] and GARA [11] projects are the foundations upon which we build our low level QoS APIs. On the contrary, research on user-level QoS has attracted less attention and left us many open issues.

The following pieces of work are directly related to our QoS-aware multimedia DBMS. In [15], 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 [16]. They argue for 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 and evaluate these cost models. A conceptual model for QoS management in multimedia database systems is introduced in [8]. 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 [8] 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 a previous work [3]. The main contribution of [3] is to specify QoS in video database queries by a query language based on constraint logic programming. They propose the *content* and *view* specifications of queries. The former addresses correctness while the latter captures the quality of the query results. Similar to [8], the paper concentrates on building a logical framework rather than the design and implementation of a real system.

Other related efforts include: The idea of *Dynamic Query Optimization* [17] is analogous to QoS renegotiation in QuaSAQ; [18] studies cost estimation of

queries under dynamic system contentions; and [19] discusses QoS control for general queries in real-time databases.

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.

References

1. L. Wolf, C. Gridwodz, and R. Steinmetz. Multimedia Communication. *Proceedings of the IEEE*, 85(12):1915–1933, December 1997.
2. H. Jiang and A. K. Elmagarmid. Spatial and Temporal Content-Based Access to Hyper Video Databases. *The VLDB Journal*, 7(4):226–238, 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. 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.
5. Klara Nahrstedt and Ralf Steinmetz. Resource Management in Networked Multimedia Systems. *IEEE Computer*, 28(5):52–63, 1995.
6. M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*, chapter 9, pages 228–273. Prentice Hall, 1999.
7. 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.
8. 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.

9. P. Seshadri. Enhanced Abstract Data Types in Object-Relational Databases. *The VLDB Journal*, 7(3):130–140, 1998.
10. 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.
11. 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.
12. H-H. Chu and K. Nahrstedt. A Soft Real Time Scheduling Server in UNIX Operating System. In *Proceedings of IDMS*, pages 153–162, 1997.
13. 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.
14. D. Yau and S. Lam. Operating System Techniques for Distributed Multimedia. *International Journal of Intelligent Systems*, 13(12):1175–1200, December 1998.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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.