Multiquality Data Replication in Multimedia Databases

Yi-Cheng Tu, *Student Member*, *IEEE*, Jingfeng Yan, *Student Member*, *IEEE*, Gang Shen, and Sunil Prabhakar, *Senior Member*, *IEEE*

Abstract—In contrast to other database applications, multimedia data can have a wide range of quality parameters, such as spatial and temporal resolution and compression format. Users can request data with specific quality requirements due to the needs of their application or the limitations of their resources. The database can support multiple qualities by converting data from the original (high) quality to another (lower) quality to support a user's query or precompute and store multiple quality replicas of data items. On-the-fly conversion of multimedia data (such as video transcoding) is very CPU intensive and can limit the level of concurrent access supported by the database. Storing all possible replicas, on the other hand, requires unacceptable increases in storage requirements. In this paper, we address the problem of multiple-quality replica selection subject to an overall storage constraint. We establish that the problem is NP-hard and provide heuristic solutions under two different system models: Hard-Quality and Soft-Quality. Under the soft-quality model, users are willing to negotiate their quality needs, as opposed to the hard-quality system wherein users will only accept the exact quality requested. Extensive simulations show that our algorithm performs significantly better than other heuristics. Our algorithms are flexible in that they can be extended to deal with changes in query pattern.

Index Terms—Quality adaptation, integer programming, data replication, heuristic algorithm.

1 Introduction

Quality is an essential property for multimedia databases. In contrast to other database applications, multimedia data can have a wide range of quality parameters, such as spatial and temporal resolution and compression format. Quality-aware multimedia systems [1], [2], [3], [4], [5] allow users to specify the *quality* of the media to be delivered based on their practical needs and resource availability on the client-side devices [2], [5]. The quality parameters of interest also differ by the type of media we deal with. For digital video, such quality parameters include resolution, frame rate, color depth, signal-to-noise ratio (SNR), audio quality, compression format, and security level [1]. For example, a video editor may request a video at very high resolution when editing it on a high-powered desktop machine, but request the video at low resolution and frame rate when viewing it using a PDA. Different encoding formats may be desirable for different applications.

From the point of view of a video database, satisfying user quality specifications can be achieved using two complementary approaches: 1) store only the highest resolution copy and convert it to the quality format requested by the user as needed at runtime or 2) precompute each different quality that can be requested and store them on a disk. When the user query is received, the appropriate copy is retrieved from the disk and sent to the user. This first approach, often called *dynamic adaptation*, suffers from a very high CPU overhead for transcoding from one quality to another [5]. Therefore, online transcoding is difficult in a multiuser environment. Our experiments (Fig. 1), run on a Solaris machine (with a 2.4 GHz Pentium 4 CPU and 1 GB of memory), confirm this claim: An MPEG1 video is transcoded at a speed of only 15 to 60 frames per second (fps) when the CPU is fully loaded by the transcoding job.[1] This corresponds to 60-240 percent of the entire CPU power if the frame rate for the video is 25 fps. We can see that CPU power is the bottleneck if we depend on online transcoding. Current state-of-the-art CPU design and transcoding techniques may increase the speed by a factor of two to four, but it still does not solve the problem. As a result, many *transcode proxy* servers or video gateways [6] with massive computing power have to be deployed. The second approach, often called *static adaptation*, attempts to solve the problem of high CPU cost of transcoding by storing precoded multiquality copies of the original media on disk. By this, the heavy demand on CPU power at runtime is alleviated. We trade disk space for runtime CPU cycles, which is a cost-effective trade-off since disks are relatively cheap.

Existing static adaptation systems are designed under one or both of the following assumptions: 1) *user requests*

- Y.-C. Tu, J. Yan, and S. Prabhakar are with the Department of Computer Sciences, Purdue University, 305 N. University Street, West Lafayette, IN 47907. E-mail: {tuyc, yanj, sunil}@cs.purdue.edu.
- G. Shen is with the Department of Statistics, Purdue University, 150 N. University Street, West Lafayette, IN 47907. E-mail: gshen@stat.purdue.edu.

Manuscript received 17 May 2006; revised 26 Sept. 2006; accepted 6 Nov. 2006; published online 24 Jan. 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0258-0506.
Digital Object Identifier no. 10.1109/TKDE.2007.1013.

1. In these experiments, we use the open-source video processing tool named *transcode*, which is available from http://www.transcoding.org.

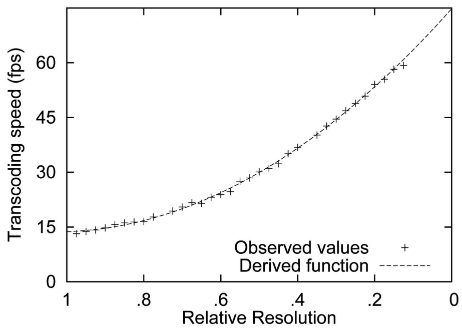

Fig. 1. Time for transcoding a 640 × 480 MPEG1 video to various (lower) resolutions.

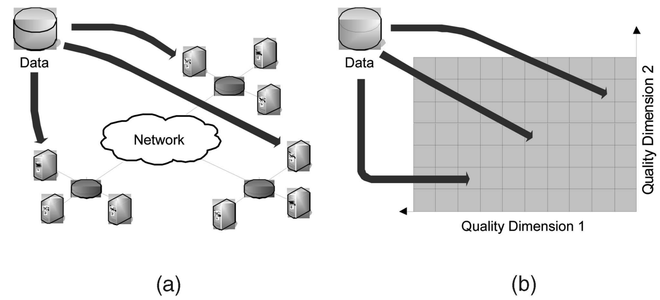

Fig. 2. (a) Traditional and (b) quality-aware data replication.

concentrate on a small number of quality profiles[2] and 2) *there is always enough storage space*. However, these are not true for real-world multimedia databases. First of all, users vary widely in their quality needs and resource availability [5]. This leads to a large number of quality-specific copies of the same media content that need to be stored on disk. Second, although cheap, storage space is not free. This is especially true for commercial media databases that must provide high reliability of disk resources (which may be leased from vendors such as Akamai). Therefore, although storage is cheap, the storage requirements should not grow unboundedly. An analysis in Section 4 shows the disk space needed to accommodate all possible qualities could be intolerably high. Therefore, the choice of which quality copies to store becomes important and is the focus of this paper.

We view the selection of media copies as a data replication problem (Fig. 2). Traditional data replication focuses on placement of copies of data in various nodes in a distributed environment [7]. Our quality-aware replication of multimedia deals with data placement in a metric space of quality values (termed *quality space*). In the traditional replication scheme, data are replicated as exact or segmental copies of the original, while the replicas in our problem are multiquality copies generated via (offline) transcoding. In this paper, we present strategies to choose the quality of replicas under two different user requirements: *Hard-Quality* and *Soft-Quality*. Under the hard-quality model, users must receive the exact quality requested. If such a quality is not already stored on disk, it must be generated by transcoding from an available quality and delivered to the client simultaneously. If the resources necessary for this transcoding are not available (e.g., due to overloading), then the request is rejected. In a soft-quality model, users are willing to negotiate the quality that they receive and may be willing to accept a quality that is close to the original request. Naturally, there is a loss in utility for the user when he has to accept a different quality, depending upon the difference in quality. In either model, a request can be rejected if the system is overloaded (at the CPU, disk, or network).

Important performance metrics for these systems include: the *reject rate* of requests, *user satisfaction*, and *resource consumption* [8], [9]. Our data replication algorithms are designed to achieve the lowest rejection rate or highest user satisfaction under fixed resource (CPU, bandwidth, and storage) capacities. We hope our work will provide useful guidelines to system designers in building cost-effective and user-friendly multimedia databases.

The remainder of this paper is organized as follows: We first compare our work with others in Section 2. We then introduce the system model in Section 3. Section 4 discusses storage use of the replication process. We present our replica selection algorithms in Sections 5, 6, and 7. Section 8 is dedicated to experimental results. We conclude the paper in Section 9.

2 Related Work and Our Contributions

A preliminary version of this paper appears in [10]. This work is motivated by the efforts to build quality-aware media systems [1], [3], [4], [5]. In [1], quality-aware query processing is studied in the context of multimedia databases. In that paper, we extend the query processing/optimization module of a multimedia DBMS to handle quality in queries as a core DBMS functionality. Two other related works in multimedia databases discuss quality specification [11] and quality model [12]. None of the above deals with replication of copies with different qualities. The closest work in quality-aware data replication is by On et al. [13]. They focus more on the availability and consistency of nonmedia data.

The traditional data caching/replication problem has been studied extensively in the context of the Web [14], [15], distributed databases [7], [16], and multimedia systems [17], [18]. The Web caching and replication problem aims at higher availability of data and load balancing at the Web servers. Similar goals are set for data replication in multimedia systems. What differs from Web caching is that disk space and I/O bandwidth are the major concerns in multimedia systems. A number of algorithms are proposed to achieve a high acceptance rate and resource utilization by balancing the use of different resources [18], [19], [20]. Unlike Web and multimedia data, database contents are accessed by both read and write operations. This leads to high requirements on data consistency, which often conflict

2. As a result of 1), many media service providers offer quality options based on the client-side devices' processing capabilities. For example, CNN.com used to provide video streaming service in three different predefined qualities: one for dial-up users, one for DSL users, and one for T1 users. However, this solution places strong limitations on the freedom of quality selection. Furthermore, with the development of mobile technologies, there are a large number of devices such as smart phones and PDAs, each of which has different rendering and communication capabilities. Thus, even by adapting this strategy, the number of possible quality-specific copies of the same media is large and keeps increasing with the emergence of new devices.

with data availability. Due to resource constraints, data consistency can sometimes only be enforced loosely.

Dynamic replication of data is another important issue since access frequencies to individual data items are likely to change in most environments. The goal is to make the replication strategy quickly and accurately adapt to changes and achieve optimal long-term performance. Wolfson et al. [21] introduced an algorithm that changes the location of replicas in response to changes of read-write patterns of data items. The interactions between query optimization and data cache utilization in distributed databases are discussed in [22]. They found that, to take advantage of cached data, it is sometimes necessary to process individual queries using "suboptimal" plans in order to reach higher system performance. In [23] and [8], video replication/dereplication is triggered as a result of changes of request rates.

Quality support in media delivery in response to heterogeneous client features and environmental conditions has attracted a lot of attention [2], [5]. The problem of quality selection under storage constraints, however, has not been well addressed due to the oversimplified assumption of unlimited storage. A work that is close in spirit to ours is presented in [24], where quality selection is performed with the goal of minimizing transcoding costs. In another related paper [25], the problem of optimal materialized view selection is studied. Both [24] and [25] address different data selection problems from ours. Furthermore, neither considers quality selection in response to dynamic changes of query pattern. Another feature of our problem is that storage is shared by multiple physical objects, each of which has it own quality space. How to distribute storage among these quality spaces brings an extra dimension of difficulty. In comparison to the preceeding papers, we make the following contributions in our study:

1. We analytically and experimentally show that the storage cost of static adaptation is so high that only a small number of replicas can be accommodated in disks.
2. In a hard-quality system where users are assumed to be strict on quality requirements, we develop a (near-optimal) replica selection algorithm that minimizes the request reject rate based on probabilistic analysis.
3. We formulate the replica selection under a soft-quality model as a facility location problem with the goal of maximizing user satisfaction. We propose a fast greedy algorithm with performance comparable to commercial optimizers. An improvement to the greedy algorithm is also discussed.
4. We extend the algorithms developed in items 2 and 3 to handle dynamic changes of query pattern. Our solutions are fast and achieve the same level of optimality as the original algorithms.

3 System Model, Notations, and Assumptions

We assume that the database consists of a collection of servers that host the media content and service user queries. For now, we consider a centralized, single server scenario. The case of multiple, distributed servers is discussed in Section 6.5.3. We list in Table 1 the notations that will be used throughout this paper.

TABLE 1
Notations and Definitions

Symbol	Definition
System parameters	
B	Server bandwidth
S	Server storage space for storing media
C	Server CPU power
V	Number of media objects in the system
P	Request reject rate
M_i	Total number of quality points for media i
M	Total number of quality points for all media
f	Total query rate, $f = \sum_{k=1}^{M} f_k$
Quality-specific parameters	
f_k	Query rate, number of requests per unit time
μ_k	Service rate, requests served per unit time
λ_k	Request intensity, $\lambda_k = f_k \mu_k^{-1}$
c_k	CPU cycles per unit time for transcoding into this quality point from the original quality
b_k	Bandwidth needed for streaming
s_k	Storage space needed if a replica is placed
P_k	Reject rate of requests to quality k

In our model, a server is characterized by the total amounts of the following resources available: bandwidth (B), storage space (S), and CPU cycles (C). Among them, *bandwidth* can be viewed as the minimum of the network bandwidth and the I/O bandwidth.

User requests identify (either directly or via a query) an object to be retrieved as well as the desired quality requirements on m quality dimensions ($\vec{q} = \{q_1, q_2, \ldots, q_m\}$, termed as *quality vector*). Each quality vector can thus be modeled as a point (hereafter called *quality point*) in an m-dimensional space. Generally, the domain of a quality parameter contains a finite number of values. For example, the spatial resolution of a video is an integer number of pixels within the range of 192 × 144 (low-quality MPEG1) to 1,920 × 1,080 (HDTV). The total number of quality points for a specific media object i is $M_i = \prod_{j=1}^{m} |Q_{ij}|$, where Q_{ij} is the set of possible values in dimension j for object i and Q_{ij} need not to be identical for all media objects. Note that every quality point is a candidate replica to be stored on disk.

Consider each possible quality, k, stored in the database. We use the following parameters to model this object: f_k, μ_k, c_k, s_k, and b_k. Quantity f_k represents the query rate for this version of the video. We assume that the query arrival is a Poisson process with this arrival rate. The query processing duration is assumed to follow an arbitrary distribution with expectation $1/\mu_k$. Note that $1/\mu_k$ may not be the same as the standard playback time of the media as the users may use VCR functionalities (e.g., stop and fast forward/backward) during media playback. The last three parameters (c_k, s_k, b_k) correspond to the usage of resources. They can be estimated from empirical functions derived by regression (see Section 4). Quantity c_k is fixed as the transcoding cost only depends on the target quality. Under the hard-quality model, the server performs the following steps upon receiving a request:

1. attempts to retrieve from disk a replica that matches the quality vector $\vec{q}$ attached to the request;

2. if the corresponding replica does not exist, it transcodes a copy from a high-quality replica (by consuming c_k units of CPU) at runtime; and
3. rejects the request if not enough CPU is available to perform item 2.

If either item 1 or 2 is performed, the retrieved/transcoded media data is transmitted to the client via the network (using b_k units of bandwidth). The request is also rejected if the required bandwidth is unavailable. We ignore the CPU costs of nontranscoding operations as they are negligible compared to transcoding costs and do not change with the specified $\vec{q}$. In the above model, requests are either admitted or rejected without waiting in a queue. The steps performed in soft-quality systems are slightly different from the above. We will discuss those in Section 6.

3.1 Assumptions

In this paper, we assume that replicas are readily available. In practice, all replicas can be precoded and archived on tertiary storage and copied into disk when a replication decision is made. We also assume that rejected queries are not reissued by the users. For analysis under the hard-quality model (Section 5), we make the following assumptions:

Assumption 1. *CPU is a heavily overloaded resource as a result of online transcoding requests, i.e.,* $\sum_{k=1}^{M} \lambda_k c_k = m_c C$, *where* $m_c \gg 1$. *On the other hand, the load put on system bandwidth is not as heavy as that on CPU, i.e.,* $\sum_{k=1}^{M} \lambda_k b_k = m_b B$ *and* $m_b \ll m_c$. *We call* m_b *and* m_c *the* load coefficients *of these resources. Note that the load on system bandwidth can be* critical, *i.e.,* $m_b = 1$ *or* light, *i.e.,* $m_b < 1$.

Assumption 2. *We assume* $\frac{c_i}{C} > \frac{b_j}{B}$, $\forall i, j$. *This means that the ratio of CPU cost to total CPU power is always higher than that of bandwidth cost to total bandwidth.*

The above two assumptions are reasonable due to our discussion in Section 1 about CPU being the bottleneck in our system model.

4 Storage Requirements for Quality-Aware Replication

As mentioned in Section 1, it is often assumed in previous works that sufficient storage is available for static adaptation. Now, we scrutinize this assumption. Since a user can request any of the possible qualities, an ideal solution is to store most, if not all, of these replicas on disk such that only minimal load is put on the CPU for transcoding. We show that the storage cost for such a solution is simply too high.

We use digital video as an example throughout this paper. According to [2], the bitrate of a video replica with a single reduced quality parameter (e.g., resolution) is expressed as:

$$F = F_0(1 - R^{\beta}), \qquad (1)$$

where F_0 is the bitrate of the original video, R is the percentage of quality change ($0 \le R \le 1$) from the original media, and β is a constant derived from experiments ($0.5 < \beta < 1$). Suppose we replicate a media into n copies with a series of quality changes R_i ($i = 1, 2, \ldots, n$) that cover the domain of R evenly (i.e., $R_i = i/n$). The sum of the bitrate of all copies is given by:

TABLE 2
Total Relative Storage in a 3D Quality Space

n	5	10	15	20	25
Storage	20.23	117.7	354.8	755.9	1496.5

$$\begin{aligned} \sum_{i=0}^{n} F_0\left(1 - R_i^{\beta}\right) &= F_0\left[n - \sum_{i=0}^{n}\left(\frac{i}{n}\right)^{\beta}\right] \\ &\approx F_0\left[n - \int_0^n \left(\frac{i}{n}\right)^{\beta} di\right] \end{aligned} \qquad (2)$$

$$\begin{aligned} &= F_0\left(n - \frac{n}{\beta+1}\right) \\ &= F_0 \frac{n\beta}{\beta+1} = F_0 O(n). \end{aligned} \qquad (3)$$

The corresponding storage requirement can be easily calculated as $TF_0\frac{n}{\beta+1}$, where T is the standard playback time of the media. Note that the above only considers one quality dimension. In [2], (1) is also extended to three dimensions (spatial resolution, temporal resolution, and SNR):

$$F = \alpha F_0\left(1 - R_a^{\beta}\right)\left(1 - R_b^{\gamma}\right)\left(1 - R_c^{\theta}\right), \qquad (4)$$

where R_a, R_b, and R_c are quality change in the three dimensions, respectively. The constants of their transcoder(s) are: $\alpha = 1.12$, $\beta = 2/3$, $\gamma = 0.588$, and $\theta = 1.0$. Using the same technique of approximation by integration as used in (2), we can easily see the sum of all storage needed for all n^3 replicas is $TF_0O(n^3)$. To be more general, the relative storage (to the original size) needed for static adaptation is on the order of total number of quality points. The latter can be represented as $O(n^d)$, where d is the number of and n is the replication density along quality dimensions. The conclusion is true as long as storage decreases polynomially with degradation of quality. Some of the storage costs generated using (4) are listed in Table 2. For example, when $n = 10$, the extra storage needed for all replicas is 117.7 times that of the original media size. No media service can afford to acquire hundreds of times of more storage for the extra feature of static adaptation. Needless to say, we could have even more quality dimensions in practice.

We have also experimentally verified the storage requirements for replication. Again, we use the *transcode* processing toolkit in these experiments. Fig. 3 shows the relative video size when spatial resolution decreases by various percentages. The discrete points are the resulting video sizes and curve A represents (1). In this graph, the areas under the curves can be viewed as the total relative storage use. We also plot a straight line B with function $1 - 1.25R$ to show the theoretical storage usage based on (3). The area of the triangle formed by the X, Y axes and line B is $\frac{2}{5}$, which is the same as that given by (3) since $\beta = \frac{2}{3}$. The fact that the areas under these three curves are almost the same corroborates our analysis in this section.

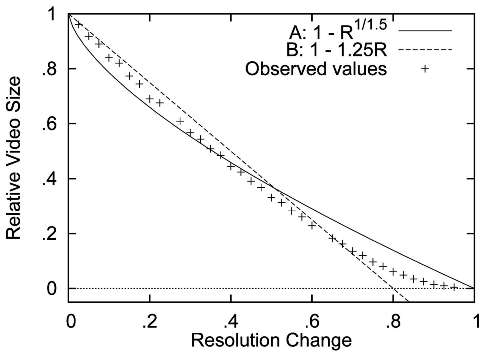

Fig. 3. Change of video size with resolution degradation.

5 Hard-Quality Systems

In this section, we discuss data replication strategies in hard-quality systems where users have rigid quality requirements on service. This means a user is not willing to negotiate when the quality she specifies cannot be satisfied. As mentioned in Section 1, the main idea of static adaptation is to replicate original media into multiple quality copies such that the demand on CPU decreases. In Section 4, we have shown that it is impractical to store all possible quality combinations. Therefore, the problem becomes how to choose quality points for replication given finite storage space C such that system performance is maximized. Since the unavailability of a requested quality results in rejection of the request, we use the *reject probability*, P, as the metric for performance evaluation. Let the output of the replica selection algorithm be a vector $(r_1, r_2, \ldots, r_M)$ with 0/1 elements ($r_k = 1$ if replica k is to be stored in disk). Formally, the replica placement problem is to

$$\begin{aligned} &\text{minimize} \quad P, \\ &\text{subject to} \quad \sum_{i=1}^{M} r_k s_k \leq S, \end{aligned}$$

where the f_k, μ_k, s_k, and c_k values for each replica are given.

To approach the above problem, it is critical to derive the relationship between P and the replica-specific values. First of all, the reject probability of all quality points is a weighted average of those of individual points:

$$P = \sum_{k=1}^{M} \frac{f_k}{\sum_{k=1}^{M} f_k} P_k = \frac{1}{f} \sum_{k=1}^{M} f_k P_k, \tag{5}$$

where P_k is the reject probability of replica k. Suppose our replication algorithm divides the M quality points into two disjoint sets: a set $\mathcal{R}$ containing replicated points and a set $\mathcal{R}'$ with nonreplicated points. Following (5), we have

$$P = \frac{1}{f}(f_{\mathcal{R}} P_{\mathcal{R}} + f_{\mathcal{R}'} P_{\mathcal{R}'}), \tag{6}$$

where $f_{\mathcal{R}} = \sum_{i \in \mathcal{R}} f_i$ is the total request rate in set $\mathcal{R}$, $P_{\mathcal{R}} = \frac{1}{f_{\mathcal{R}}} \sum_{k \in \mathcal{R}} f_k P_k$ is the reject probability of all requests from $\mathcal{R}$, and $f_{\mathcal{R}'}$, $P_{\mathcal{R}'}$ are the counterparts of $f_{\mathcal{R}}$, $P_{\mathcal{R}}$ in set $\mathcal{R}'$.

In our model, the admission of a request is determined by the runtime availability of two resources: bandwidth and CPU. If either is insufficient to serve the request, the request is rejected. So, the reject probability for a set of objects, say, those in set $\mathcal{R}$, can be expressed as

$$P_{\mathcal{R}} = P_{\mathcal{R}}^{(b)} + P_{\mathcal{R}}^{(c)} - P_{\mathcal{R}}^{(bc)} \tag{7}$$

where $P_{\mathcal{R}}^{(b)}$, $P_{\mathcal{R}}^{(c)}$, and $P_{\mathcal{R}}^{(bc)}$ are probabilities of the following events happening to set $\mathcal{R}$ requests: *rejected by bandwidth*, *rejected by CPU*, and *rejected by both CPU and bandwidth*. Note we cannot say $P_{\mathcal{R}}^{(bc)} = P_{\mathcal{R}}^{(b)} \cdot P_{\mathcal{R}}^{(c)}$ as the first two events could be dependent on each other. Similarly, we have the following for set $\mathcal{R}'$:

$$P_{\mathcal{R}'} = P_{\mathcal{R}'}^{(b)} + P_{\mathcal{R}'}^{(c)} - P_{\mathcal{R}'}^{(bc)}, \tag{8}$$

where $P_{\mathcal{R}'}^{(b)}$, $P_{\mathcal{R}'}^{(c)}$, and $P_{\mathcal{R}'}^{(bc)}$ are defined according to set $\mathcal{R}'$ requests.

As no rejection by CPU will occur when a replica is placed in disk (Section 3.1), we have $P_{\mathcal{R}}^{(c)} = 0$, which leads to $P_{\mathcal{R}}^{(bc)} = 0$ and, thus, $P_{\mathcal{R}} = P_{\mathcal{R}}^{(b)}$. Plugging this and (8) into (6), we have

$$P = \frac{1}{f}\left[f_{\mathcal{R}} P_{\mathcal{R}}^{(b)} + f_{\mathcal{R}'}\left(P_{\mathcal{R}'}^{(b)} + P_{\mathcal{R}'}^{(c)} - P_{\mathcal{R}'}^{(bc)}\right)\right]. \tag{9}$$

We now establish the following proposition and theorem that will help analyze the above expression. Although both Proposition 1 and Theorem 1 seem intuitively obvious, their proofs are nontrivial extensions of well-established results in queuing theory [26]. The basic idea is to map the hard-quality system to an Erlang loss model: We can view the bandwidth (CPU) as a resource pool with B (C) channels, the replica-specific requests are modeled as Poisson streams with arrival rate f_k, and service rate μ_k, and no waiting queue exists (i.e., lossy system). What complicates our analysis is that each request class requires a different number of channels (i.e., c_k, b_k), whereas exactly one channel is used for one request in a regular Erlang system (e.g., one line for each telephone call). It is reported that stationary distributions do exist for the reject probability in such systems. The main results of such studies can be found in Appendix A.

Proposition 1. *Given two Erlang loss systems $\mathcal{A}$ and $\mathcal{B}$, each has a number of traffic classes. Any class i in group $\mathcal{A}$ is characterized by its arrival rate f_i, service time μ_i, and number of channels needed for each connection c_i. Similarly, a class j in group $\mathcal{B}$ is by f_j, μ_j, and b_j. Let R_A and R_B be the total number of channels for system $\mathcal{A}$ and $\mathcal{B}$, respectively. If 1) $R_A > R_B$, 2) $m_A \gg m_B$, where m_A and m_B are the load coefficients of systems $\mathcal{A}$ and $\mathcal{B}$ (i.e., Assumption 1 in Section 3.1), and 3) $\min_{i \in A}\{c_i\}/R_A > \max_{j \in B}\{b_j\}/R_B$ (i.e., Assumption 2 in Section 3.1), then the reject probability in system $\mathcal{A}$ is greater than that of system $\mathcal{B}$.*

Proof. See Appendix B. □

Theorem 1. *If the requested load on a resource in the hard-quality system is* critical *or* light, *we have*

$$P = O\left(\sqrt{\frac{2\beta}{N\pi f^2}}\right),$$

where N is the scale of resource pool (i.e., $N = \Theta(B)$ for bandwidth and $N = \Theta(C)$ for CPU) and $\beta = \sum_{i=1}^{M} f_k \mu_k$.

Proof. See Appendix C. □

The nonreplicated set $\mathcal{R}'$ and replicated set $\mathcal{R}$ can be mapped to groups $\mathcal{A}$ and $\mathcal{B}$ in Proposition 1, respectively. Since storage is limited, replicating some qualities does not change the fact that CPU is still heavily loaded (i.e., condition 2 always holds). Thus, we have $P^{(b)}_{\mathcal{R}} < P^{(c)}_{\mathcal{R}'} \leq P_{\mathcal{R}'}$ when both resources are overloaded. This also holds true when the bandwidth (i.e., group $\mathcal{B}$) has critical or light load because the reject probability is close to zero under such conditions (Theorem 1, it is easy to see that $\beta < f^2$ and N are large in our problem). The second inequality in the preceding formula is given by the fact that $P^{(b)}_{\mathcal{R}'} \geq P^{(bc)}_{\mathcal{R}'}$.

Revisiting (9), as $P^{(b)}_{\mathcal{R}} < P_{\mathcal{R}'}$, no matter how we choose members of $\mathcal{R}$ and $\mathcal{R}'$, a heuristic solution to the problem of minimizing P would be to maximize $f_{\mathcal{R}}$ (or minimize $f_{\mathcal{R}'}$ since $f_{\mathcal{R}} + f_{\mathcal{R}'} = f$) subject to $\sum_{k\in\mathcal{R}} s_k \leq S$. In other words, the problem becomes the classic 0-1 Knapsack problem, which can, in turn, be solved by the following heuristic algorithm: We sort all possible qualities by their request rate per unit size (f_k/s_k) and select those with the highest such values until the total storage is filled. The running time of this algorithm is $O(M \log M)$. In Appendix D, we show that the results obtained by such a heuristic are near-optimal when $S \gg s_k$ for all $k \in [1, M]$, which is a safe assumption. The above result is interesting in that it shows that f_k and s_k are the only factors we need to consider in quality selection even though the reject probability is also a function of μ_k, c_k, and b_k (Appendix A).

6 Soft-Quality Systems

In hard-quality systems, replicas of the same media are treated as independent entities: Storing a replica with quality $\vec{q_1}$ does not help the requests to another with quality $\vec{q_2}$ as quality requirements are either strictly satisfied or the request is rejected. However, users can generally tolerate some changes of quality [2] and the parameters specified only represent the most desirable quality. If these parameters cannot be exactly matched by the server, they are willing to accept a similar set of qualities. The process of settling down to a new set of quality parameters is called *renegotiation*. Of course, the deviation of the actual qualities a user gets from those he/she desires will have some impact on the user's viewing experience and the system should be penalized for that.

6.1 Utility Functions

We generally use *utility* to quantify user satisfaction on a service received [27]. For our purposes, *utility functions* can be used to map quality to utility and the penalty applied to the media service due to renegotiation is easily captured by *utility loss*. As utility directly reflects the level of satisfaction from users, it is the primary optimization goal in quality-critical applications [9]. We thus set the goal of our replica selection strategies to be maximizing utility. The server operations shown in Section 3 need to be modified in soft-quality systems. For simplicity, we assume the "renegotiation" process between the client/server is instantaneously performed on the server side based on a simple rule: In case of a miss in Step 1, the server always chooses a replica that yields the largest utility for the request to retrieve.

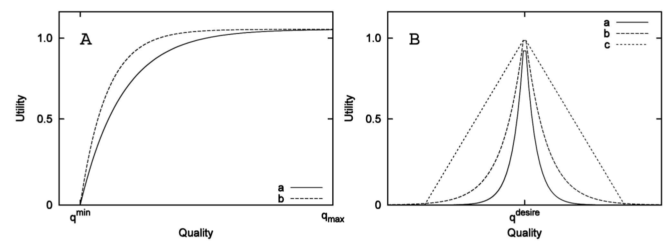

Fig. 4. Different types of utility functions.

Fig. 4 shows various types of utility functions for a single quality dimension. In general, utility functions are convex monotones (Fig. 4A) due to the fact that users are always happy to get a high quality service, even if the quality exceeds his/her needs [27]. However, in a more realistic environment, the cost of the extra quality may be high as more resources have to be consumed (Section 1) on the client side. Thus, excessively high quality negatively affects utility. Taking this into account, we propose a new group of utility functions in quality-aware media services: It achieves the maximal utility at a single point q^{desire} and monotonically decreases on both sides of q^{desire} along the quality dimension (Fig. 4B). Note that the functions do not have to be symmetric on both sides of q^{desire}. The hard-quality model in Section 5 can be viewed as a special case: Its utility function takes the value of 1 at q^{desire} and 0 otherwise. The utility for a quality vector with multiple dimensions is generally given as a weighted sum of the dimensional utility [9].

6.2 Data Replication as an Optimization

In this subsection, we formally define the replica selection problem in soft-quality systems. Let us first study how to choose replicas for one media object i. We then extend our discussion to all V objects in Section 6.5.1.

The problem is to pick a set of L replicas that gives the largest total *utility* over time, which can be expressed as

$$U = \sum\nolimits_{j\in J} f_j u(j, L),$$

where J is the set of all M_i points and $u(j, L)$ is the largest utility with which a replica in L serves a request for quality j. Obviously, $u(j, L)$ has maximum value when $j \in L$. We set $u(j, L)$ to be a function of the distance between j and its nearest neighbor in L (Section 6.1). Generally, $u(j, L)$ is normalized into a value in [0, 1]. We weight the utility by the request rate f_j and the weighted utility is termed as *utility rate*. The constraint of forming set L is that the total storage of all members of L cannot exceed S. We name our problem the *fixed-storage replica selection* (FSRS) problem, which can be formulated as the following integer programming:

$$\text{maximize} \quad \sum\nolimits_{j\in J} \sum\nolimits_{k\in J} f_j u(j, k) Y_{jk}, \tag{10}$$

$$\text{subject to} \quad \sum\nolimits_{k\in J} X_k s_k \leq S, \tag{11}$$

$$\sum\nolimits_{k \in J} Y_{jk} = 1, \quad (12)$$

$$Y_{jk} \leq X_k, \quad (13)$$

$$Y_{jk} \in \{0, 1\}, \quad (14)$$

$$X_k \in \{0, 1\}, \quad (15)$$

where $u(j, k)$ is the utility value when a request to point k is served by a replica in j, X_k is a binary variable representing whether k is replicated, Y_{jk} tells if j should be served by k. Equation (11) shows the storage constraint, while (12) and (13) mean all requests from k should be served by one and only one replica.

The FSRS problem looks similar to a group of well-studied optimizations known as the *uncapacitated facility location* (UFL) problems [28]. Yet, it is different from any known UFL problems in that the storage constraint in FSRS is unique. A close match to FSRS is the so-called p-median problem with the same problem statements, except (11) becomes $\sum X_k = p$, meaning only p $(p < |J|)$ facilities are to be built. As the p-median problem is NP-hard [29], we can thus conclude FSRS is also NP-hard.

Theorem 2. *The FSRS problem is NP-hard.*

Proof. The p-median problem is equivalent to finding the set of replicas that yields the smallest loss of utility rate in a quality space where $s_k = \delta$ $(\delta > 0)$ for all k and $S = p\delta$. Thus, the p-median problem is polynomial time reducible to the FSRS problem and this concludes the proof. □

6.3 The Greedy Algorithm

As in the Knapsack problem, we can use a benefit/cost model to evaluate a replica k: The cost is obviously the storage s_k, the benefit would be the gain of utility rate of selecting k. What complicates the problem is that the benefit is not fixed: It depends on the selection of other replicas. To bypass this difficulty, we propose an algorithm (we call it *Greedy*) that takes guesses on such benefits. The main idea is to aggressively select replicas one by one. The first replica is assigned to a point k that yields the largest $\Delta U_k / s_k$ value as if only one replica is to be placed. We denote $\Delta U_k / s_k$ as the *utility density* of replica k, where ΔU_k is the marginal utility rate gained by replicating k. The following replicas are determined in the same way with the knowledge of replicas that are already selected. The utility density value represents our guess of the benefit-to-cost ratio in replicating k.

Fig. 5 shows the pseudocode of the *Greedy* algorithm. GREEDY calls ADD-REPLICA continuously, with a queue *list* holding the replicas selected so far. The algorithm terminates when no more replicas can be added due to storage constraints. The subroutine ADD-REPLICA is the core of this algorithm: It selects a new replica given those chosen in *slist*. It does so by trying all M_i points in the quality space (line 2) to look for the one that yields the largest utility rate. Subroutine MAXUTILITY gives the utility from j to its nearest replica in $slist + k$, which can be done in constant time if we store the previous nearest replica for all j. The two loops both have to run M_i iterations; therefore, the time complexity for *Greedy* is $O(IM_i^2)$ for one media i. Here, I is the number of replicas eventually placed in *list*. In the worst case, when all points are selected, we have $I = M_i$.

```
Algorithm GREEDY
Inputs: f_k, s_k (∀k), and S
Output: a set of selected replicas, P
1  s' ← S, P ← ∅, k ← 0
2  while k ≠ NULL do
3        k ← ADDREPLICA(s', P)
4        s' ← s' − s_k
5        append k to P
6  return P

ADDREPLICA (s, P')
1  i ← NULL, V_max ← 0
2  for each quality point k do
3      if k ∉ P' and s_k ≤ s
4        U ← 0
5        for each quality point j
6            U ← U + MAXUTIL(j, k, P')
7        if U/s_k > V_max
8          V_max ← U/s_k
9          i ← k
10 return i
```

Fig. 5. The *Greedy* algorithm.

Effects of the type of utility functions. It is easy to see that the shape of the utility functions affects the final results of replica selection. Recall that we evaluate a replica k by its $\sum f_j u(j, k)/s_k$ value, where j are the points k serves (line 7 in ADD-REPLICA). If the utility drops very fast, a replica can only collect utility from points that are extremely close to it; therefore, the *Greedy* algorithm favors those with high query rates in their close neighborhood. On the other hand, if utility drops very slowly, we may overestimate the utility rate of a point at early stages of *Greedy*. As a result, the first few replicas chosen by *Greedy* tend to be those with small s_k values since the utility rates of all candidates have little difference at that moment. In Section 6.4, we propose a solution to remedy this problem of *Greedy*.

The utility curves we have discussed so far are all monotonically decreasing functions of distance (between two points). However, our FSRS algorithm does not depend on any special features (e.g., monotonicity and convexity) of the utility functions. In fact, *Greedy* works for arbitrary types of utility functions as long as the utility value between two points is not affected by the replica selection process.

6.4 The Iterative Greedy Algorithm

The *Iterative Greedy* algorithm attempts to improve the performance of *Greedy*. We notice that, at each step of *Greedy*, some local optimization is achieved: The $(K+1)$th replica chosen *is* the best given the first K replicas. The problem is: We do not know if the first K replicas are good choices. However, we believe the $(K+1)$th replica added is more "reliable" than its predecessors because more global information (i.e., other selected replicas) is leveraged in its selection. Based on this conjecture, we develop the *Iterative Greedy* algorithm that iteratively improves the "correctness" of the replicas chosen. Specifically, we repeatedly get rid of the most "unreliable" selected replica and choose a new one. Note that the one that is eliminated is also a candidate of the selection process.

The operations in *Iterative Greedy* are shown in Fig. 6. All replicas selected by *Greedy* are stored in a FIFO queue *slist*. In each iteration, we dequeue *slist* and find one replica

```
Algorithm ITERATIVEGREEDY
Inputs: selected replicas slist, number of iterations I
Output: a modified list of replicas newlist
1  copy slist to newlist
2  U_max ← 0, storage ← available storage
3  for i ← 0 to I
4      do k ← head of slist
5          storage ← storage + s_k
6          l ← ADD-REPLICA(storage, slist)
7          append l to slist
8          update storage, U ← total utility of slist
9          if U_max < U
10             U_max ← U
11             copy slist to newlist
12 return newlist
```

Fig. 6. The *Iterative greedy* algorithm.

(line 6) among the remaining replicas. The newly identified replica is then added to the tail of *slist*. The same subroutine, ADD-REPLICA, is used to find new replicas. We keep dequeuing *slist* and running ADD-REPLICA until I iterations are finished. We record the set of replicas with the largest utility rate as the final output. As ADD-REPLICA runs in $O(M_i^2)$ time, *Iterative Greedy* has time complexity of $O(IM_i^2)$, which is the same as that of *Greedy*. The only problem here is how to set the number of iterations I. Since the primary goal of *Iterative Greedy* is to reconsider the selection of the first few "unreliable" replicas, we can set I to be smaller than the total number of replicas selected by *Greedy*.

6.5 Other Issues

6.5.1 Handling Multiple Media Objects

With very few modifications, both the *Greedy* and the *Iterative Greedy* algorithms can handle multiple media objects. The idea is to view the collection of V physical media as replicas of one virtual data object. The different content in the physical media can be modeled as a new quality dimension called *content*. A special feature of *content* is its lack of adaptability, i.e., any replica of the movie *Matrix* cannot be used to serve a request to the movie *Shrek*. Assume all physical media have a quality space with $\hat{M}$ points, the FSRS problem with V media can be solved by simply running the *Greedy* algorithm for the virtual media with $V\hat{M}$ points. Knowing that there is no utility gain between two replicas with different *content*, we only need to run the second loop (line 5) in ADD-REPLICA for those with the same content. Thus, the time complexity of GREEDY becomes $O(IV\hat{M}^2)$.

6.5.2 Relaxing/Tightening Constraints

Another point is that we set a constraint of replicating at least one copy for the video in (12). In the multi-object scenario, we can further relax or tighten this constraint. To relax it, we allow no replica being selected for a video by modifying (12) to $\sum_{k \in J} Y_{jk} \leq 1$. This requires no changes to our algorithms. On the other hand, the system administrator could also enforce the selection of certain replicas (e.g., the original video). Again, our FSRS algorithms can easily handle this: We just start running ADD-REPLICA with a list of all replicas that *must* stay in storage. However, if we stick to the original constraint (i.e., (12)) but do not specify which replica to store for each video, the problem becomes trickier as our algorithms may assign no replica to videos with low query rates. The solution is to start by selecting the smallest replica for all videos and run *Greedy*. This guarantees one replica for each video, but the effects of the constraint are minimized. Unless specified otherwise, the following extensions are based on a multivideo environment with the relaxed constraint.

6.5.3 Distributed Data Replication

In Sections 5 and 6, we discussed the strategies of quality-aware data replication in a single server. Now, we extend the solutions to a distributed system with multiple servers. Let us first investigate how the problem is changed when we consider multiple servers in a hard-quality system. Here, we assume user requests can be served by any one of $n > 1$ servers.[3] As we can see, the analysis we show in Section 4 still holds true and we can use (9) to guide our replica selection: The strategy is, again, to maximize $f_{\mathcal{R}}$. When we obtain a set of replicas with the largest possible f_A, how to assign these replicas to n servers becomes a problem. We can immediately relate this to a load balancing problem with the goal of achieving uniform reject probability in all servers. A more detailed justification and a solution can be found in [30].

7 Dynamic Data Replication

So far, we have considered the situation of *static* data replication, in which access rates of all qualities do not change over time. The importance of studying static replication can be justified by two observations: 1) Access patterns to many media systems, especially video-on-demand systems, remain the same within a period of at least 24 hours [17]. 2) Conclusions drawn from static replication studies form the basis of dynamic replication research [23]. In this section, we discuss quality-aware data replication in an environment where access patterns change. There are two main requirements to a dynamic replication scheme: quick response to changes and optimality of results. Our goal is to design real-time algorithms that match static replication algorithms in result optimality.

In this section, we focus on dynamic replication in soft-quality systems. Our solution to hard-quality systems can easily be made adaptable to dynamic situations (see [30] for more details): The replication decision is made by sorting replicas by their $\eta_k = f_k/s_k$ values. When the query rate of a replica changes, we just reinsert the replica into the sorted list and make decisions based on its current position in the list. The time complexity of this algorithm is $O(\log M)$.

Dynamic replication in soft-quality systems is a very challenging task. The difficulty comes from the fact that the access rate change of a single point could have cascading effects on the choice of many (if not all) replicas. We may have to rerun the static algorithms (e.g., *Greedy*) in response to such changes but these algorithms are too slow to make online decisions. Fortunately, the *Greedy* and *Iterative Greedy* algorithms we developed have properties that we can

3. If each server only handles requests from its local region, the problem is trivial as we only need to perform single-server replication at each server.

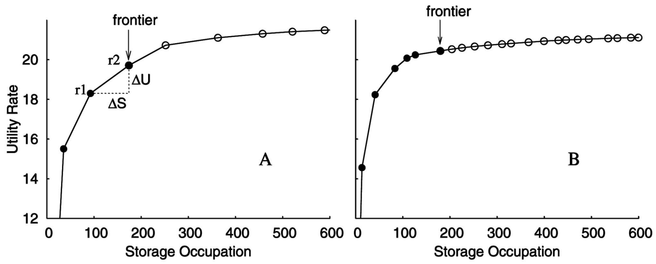

Fig. 7. Replication roadmaps.

exploit in building efficient, accurate dynamic replication algorithms. In this section, we assume that runtime variations of access pattern only exist at the media object level. In other words, the relative popularities of different quality points for the same media object do not change. Although this assumption is reasonable in many systems [17], [18], we understand that a solution for a more general situation is meaningful and we leave it as future work.

7.1 Replication Roadmap

Let us first investigate how ADD-REPLICA, being the core of both *Greedy* and *Iterative Greedy*, selects replicas. The history of total utility rate gained and storage spent on each selected replica can be represented as a series of points in a 2D graph. We call the lines that connect these points in the order of their being selected a *Replication Roadmap* (RR). Fig. 7 shows two examples of RRs plotted with the same scale. We can see that any RR is convex. The reason for this is: The slope of the line connecting any two consecutive points (e.g., $r1$ and $r2$ in Fig. 7) in an RR represents the ratio of ΔU_{r2} to s_{r2}. As ADD-REPLICA always chooses a replica with the largest $\Delta U/s$ value, the slopes of the lines along the RR are thus nonincreasing.

We can also draw RRs for individual media objects. It is not hard to see that single-media RRs are also convex. In dynamic replication, replicas need to be reselected with respect to the new query rate of a media object. Suppose the query rate f_i of a medium i increases by a factor δ ($\delta > 0$). This makes the slopes of all pieces in i's RR increase by δ. What happens now is that we may consider assigning extra storage to i as it reaches a position to use storage more profitably than before. As storage is limited, the extra chunk should come from another medium whose slope in the last piece of RR is small. Take Fig. 7 as an example. Suppose we have fully extended RRs: All future replicas are precomputed (empty dots in Fig. 7) and we call the last real replica the *frontier* of the RR. It buys us more utility to advance A's frontier (take storage) and move backward on B's RR (give up storage). The beauty of this scheme is that we never need to pick up points far into or over the frontier to make storage exchanges. The convexity of RRs tells us that the frontier is always the most efficient point to acquire/release storage. Based on this idea, we have the an online algorithm named SOFTDYNAREP for dynamic replication (Fig. 8).

7.2 The SOFTDYNAREP Algorithm

The algorithm consists of two phases: the *Preprocess Phase* and the *Online Phase*. In the Preprocess phase, we need to

```
Algorithm SOFTDYNAREP
Preprocess Phase
1  run GREEDY or Iterative Greedy
2  for all V media objects
3       store the post-frontier segment of the RR in flist
4       store the pre-frontier segment of the RR in blist
5       extend RR to its full length
Online Phase
6  Case 1. f_i increases
7   recalculate slopes of stored segments for i
8   update blist and flist (reinsertion)
9   do STORAGEEXCHANGE
10 Case 2. f_i decreases, symmetric to Case 1

STORAGEEXCHANGE
1  storage ← available storage
2  k ← 0, j ← V − 1
3  while k ≤ 0
4  do     r_0 ← flist[k]
5         victims ← ∅
6         while storage < size of replica r_0
7         do     r_1 ← blist[j]
8                if r_0 and r_1 belong to the same video
9                   j ← j − 1
10                  continue
11               if utility density of r_1 > utility density of r_0
12                  k ← k + 1
13                  rollback blist to its status on line 6
14                  break
15               else append r_1 to victims
16                    update and sort blist
17        if storage ≥ size of replica r_0
18           EXCHANGE (r_0, victims)
19           update and sort both flist and blist
```

Fig. 8. Soft-quality dynamic replication algorithm.

extend each RR formed by *Greedy* or *Iterative Greedy* by adding all M_i replicas.[4] For all RRs, we put the immediate predecessor of the *frontier* in a list called *blist* and the immediate successor in a list called *flist*. Both lists are sorted by the slopes of the segments stored. The *Preprocess phase* runs at $O(V\hat{M}^3)$ time and it only needs to be executed once.

The *Online Phase* is triggered once we detect a change in query rate to an object i. The idea is to iteratively take storage from the end of *blist* until a new equilibrium is reached. The running time of this phase is $O(I \log V)$, where I is the number of storage exchanges (line 9). In the worst case, where most of i's replicas are to be stored, we have $I = O(M_i)$. The case of query rate decrease is just handled in the opposite way of what we discussed above. In EXCHANGESTORAGE, there are two loops: In the outer loop (line 3), we choose the replica (r_0) on the head of $flist$ and try to find a list $(victims)$ of replicas on the tail of $blist$ from where storage can be taken via the inner loop (line 6). The list $victims$ has to be formed as the size of r_0 can be larger than that of one single victim replica r_1. The subroutine EXCHANGE basically dereplicates those in $victims$ and replicate r_0. The inner loop terminates when enough storage is found for r_0 or we reach a replica whose utility density is greater than that of r_0 (line 11). The latter case also terminates the outer loop (as $k > 0$).

4. In practice, we do not have to extend an RR to its full length if we can bound the possible changes of query rates.

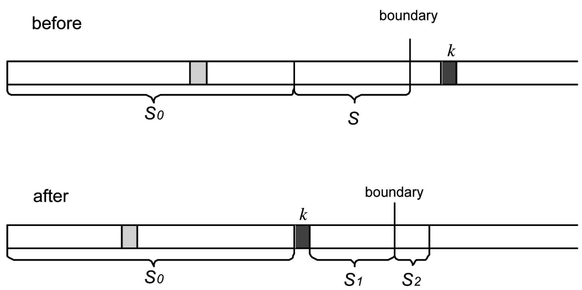

Fig. 9. Replica selection upon query rate change.

7.3 Optimality of SOFTDYNAREP

In this section, we show that the online phase of SOFTDYNAREP achieves (almost) the same quality in the selected replicas as that by rerunning GREEDY at runtime.

From discussions in Section 7.1, we know that the global RR changes as the query rates of individual replicas change and GREEDY (implicitly) rebuilds the whole global RR. Essentially, *Greedy* selects those replicas with the largest utility density on the global RR, similarly to our solution to the hard-quality problem (i.e., a 0-1 Knapsack). Let us consider a modified version of *Greedy* named M-GREEDY, which has a subtle difference from the original GREEDY algorithm. In M-GREEDY, we replicate items along the global RR until we encounter the first replica k' that cannot be accommodated by the available storage (equivalent to l in Appendix D).[5] We immediately see that GREEDY is smarter than M-GREEDY: It will try to fill the available storage with replicas with lower utility density than k'. Thus, the replicas selected by M-GREEDY are prefixes of the global RR (from the beginning to the one prior to k'), while those selected by GREEDY are not a consecutive chunk in the RR. Due to the same reasons discussed in Appendix D, the total utility rate achieved by M-GREEDY is only trivially smaller than that of GREEDY. To prove our claim that SOFTDYNAREP is as good as GREEDY, we have the following lemma:

Lemma 1. *With the same replica-specific inputs and change of query rate of a specific video, if a replica is selected by M-GREEDY, it is also selected by SOFTDYNAREP.*

Proof. Let us first study the change of the global RR before and after the query rate change. In Fig. 9, the global RR is represented as an array of replicas sorted by descending order of utility density. We know that GREEDY selects replicas from the left to the right until no storage is available. We draw a line called *boundary* between those that are replicated and those that are not. We consider the case of query rate increase of an object v. As a result of the query rate increase, some replicas of v (represented as shaded boxes in Fig. 9) will move toward the left in the array of replicas and a new boundary will be formed. However, the relative order of all replicas of v does not change. Therefore, there are two types of selected replicas by the M-GREEDY algorithm after the change: 1) those that were not selected before the change, and 2) those that were selected before the change. We prove SOFTDYNAREP selects the corresponding replicas in both cases.

Case 1. Without loss of generality, we consider a replica k of v that moves across the boundary in M-GREEDY. The selection of k can be achieved by one of two means: 1) The storage left before the change is greater than s_k and 2) storage is taken from replicas with utility density is smaller than that of k. It is easy to see that k will be the head of $flist$ in SOFTDYNAREP. In the former case, we go directly to line 17 of STORAGEEXCHANGE (Fig. 8) and replicate k. For the second situation, a list of replicas are chosen to give up their storage to k (loop in line 6). As long as there is enough storage from those with smaller utility density, k will be replicated.

Case 2. The replicas considered in this case can be divided into two categories:

Case 2.1. Replicas whose utility density is greater than that of k (e.g., those in region S_0 in Fig. 9). These replicas are not affected by SOFTDYNAREP as we never sacrifice such replicas for k (line 11 of STORAGEEXCHANGE).

Case 2.2. Replicas whose utility density is smaller than that of k (e.g., those in region S_1 in Fig. 9). These replicas are part of region S before the query rate changes. One feature of M-GREEDY is that all replicas chosen form a consecutive chunk in the list. To accommodate k, S is simply cut into two consecutive regions S_1 and S_2. In SOFTDYNAREP, the same list *victims* is also a consecutive chunk as it is formed by always choosing the replica with the smallest utility density, starting (backward) from the end of S. Furthermore, it ends as long as enough storage is found; thus, everything in S_1 will not be included in $victims$.

The case of multiple replicas crossing the boundary and decrease of query rate would not complicate the above argument. □

8 EXPERIMENTS

We study the various algorithms described in previous sections by extensive simulations. We use 270 MPEG1/2 videos stored in a real video database as experimental data.[6] For all replicas, we set their μ_k^{-1} values to be their standard playback time. The videos are then transcoded into replicas of different spatial resolution and frame rates using *transcode* (Section 4) to generate the b_k, c_k, and s_k values for all replicas. We test various access patterns (e.g., uniform, Zipf, 20-80, 10-90) in our simulations. The simulated video servers possess network bandwidth of 90 Mbps (dual T3 lines), four UltraSparc 1.2 MHz CPUs, and variable storage capacity (60 to 300 G) for data replication. All of the above parameters are set to be close to those in a real-world server[7] and we simulate a cluster of 10 such servers. We perform our simulations on a Sun Workstation with a UltraSparc 1.2 MHz CPU and 2 GB of main memory running Solaris 8.

5. We use M-GREEDY as a conceptual variance to GREEDY, its implementation is irrelevant to our discussions.

6. http://www.cs.purdue.edu/vdbms.

7. The total storage is relatively small because we only have 270 raw videos in the simulated system and the quality of most of the videos are not very high. In real systems, storage is more abundant, but we may also have much more raw media with higher quality.

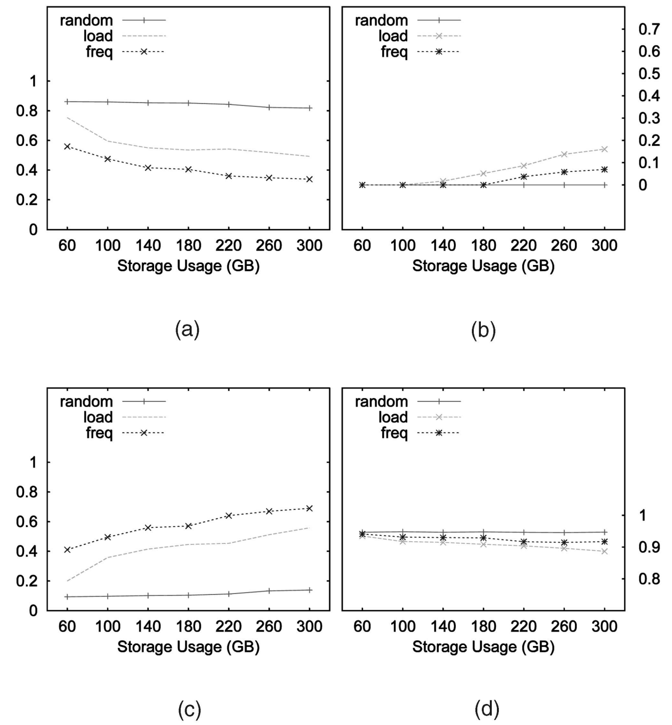

Fig. 10. Performance of various replica selection algorithms in the hard-quality model. (a) P. (b) $P_R^{(b)}$. (c) f_R/f. (d) $P_R^{(c)}$.

8.1 Results for Hard-Quality Systems

In this experiment, we compare our replica selection algorithm (Section 5) to various heuristics under the hard quality system model. The metric is the reject frequency measured as the ratio of the total number of rejected requests to total requests. The quality space is a 2D space (resolution and frame rate) with 15 to 20 values on each dimension (differs by each video object). Requests (with $f = 7,200/hour$) are distributed in a Zipf pattern to all M replicas.

In Fig. 10, we show the performance of three quality selection methods: 1) our solution that chooses quality points by their f_k/s_k values ("*freq*"), 2) an algorithm that randomly chooses replicas one by one till all storage is filled ("*random*"), and 3) an algorithm ("*load*") that places the largest possible load into set $\mathcal{R}$ by choosing qualities with the largest CPU load to storage ratio ($\frac{\lambda_k c_k}{s_k}$). The results confirm our analysis in Section 5 as our solution (*freq*) always gets the lowest reject probability (Fig. 10a). As expected, the total request rate in set $\mathcal{R}$ achieved by *freq* is always the highest (Fig. 10b). In fact, the $f_\mathcal{R}$ value achieved in our results is very close to those given as the upper bound of the optimal such values in all cases.[8]

From Fig. 10c and Fig. 10d, we can see that the rejection frequency on bandwidth is significantly smaller than that on CPU. In these experiments, the recorded load coefficient of bandwidth (m_b) range from 0.24 to 1.26. On the other hand, the load coefficients of CPU rendered by the same jobs range from 16 to 36. This explains why the observed reject frequency on CPU (Fig. 10d) is always high (> 0.88). For algorithms *freq* and *load*, as storage increases, P and $P_{\mathcal{R}'}^{(c)}$ decrease while $P_\mathcal{R}^{(b)}$ and $f_\mathcal{R}$ increase. Note that, when excessive storage is used, the decrease of P slows down as congestion on bandwidth becomes more significant. The performance of *random* is not affected by total storage.

8. The upper bound is described in Appendix D and is not plotted here.

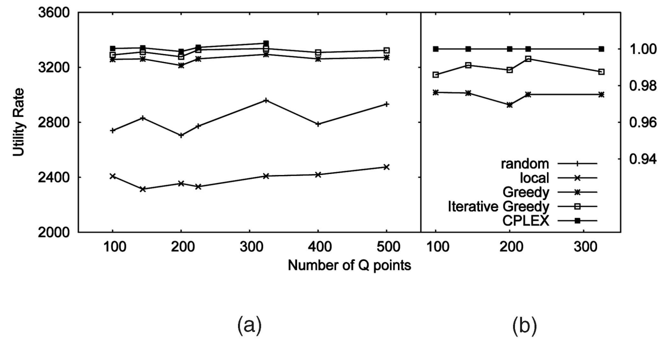

Fig. 11. Optimality of replica selection algorithms. (a) Absolute utility rate. (b) Relative U to optimal.

8.2 Results for Soft-Quality Systems

In this section, we present experimental results under the soft-quality model. We first evaluate the performance of the GREEDY and ITERATIVEGREEDY algorithms in terms of optimality (Fig. 11) and running time (Fig. 12). In this experiment, we set f to 3,600 requests/hour; thus, the utility rate is bounded by 3,600/hr. We compare our algorithms with three others: 1) the CPEX mathematical programming package.[9] CPLEX is a widely used software for solving various optimization problems and is well known for its efficiency. We tune CPLEX such that the utility rate of results obtained are within a 0.01 percent gap to that of the optimal solution. 2) The same *random* algorithm as the one described in Section 8.1. 3) A *local* algorithm that places replicas in the hottest areas in the quality space.[10] We run the experiments for a total of 30 media objects.[11] Each data point represents the mean of four simulations.

From Fig. 11a, it is clear that our algorithms always find solutions that are very close to the optimal. More details can be found in Fig. 11b, where the relative U values obtained by our algorithms to those by CPLEX are plotted. Utility rates of solutions found by GREEDY are only about 3 percent smaller than the optimal values. The ITERATIVE GREEDY cuts the gap by at least half in all cases: Its solutions always achieve more than 99 percent of the optimal utility rate. The performance of neither algorithm is affected by the increase in number of quality points. Nor is it affected by access patterns: We tested different access patterns (e.g., Zipf, 20-80, 10-90, and uniform) and obtained similar results (data not plotted). The solutions given by *random* and *local* are far from optimal. Surprisingly, the *local* algorithm, which is similar to our solution under the hard-quality model (Section 5), performs even worse than the random algorithm. This reiterates that it is dangerous to

9. version 8.0.1, http://www.cplex.com.

10. Specifically, we divide the whole space into squares of equal size. Each square is evaluated by the sum of the query rates of all quality points it contains. We place a replica in the center of those squares with the largest total query rates until all storage is filled.

11. Here, we choose a small number of media so that it is feasible to find the optimal solutions as a comparison to our solutions.

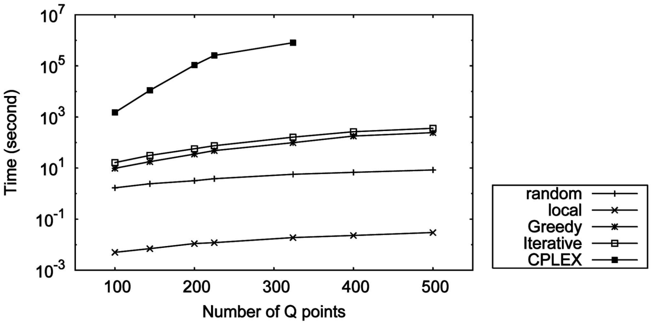

Fig. 12. Running time of different replica selection algorithms.

consider only local or regional information in solving a combinatorial problem.

The running time of the above experiments are shown on a logarithmic scale in Fig. 12. CPLEX is the slowest algorithm in all cases. This is what we expected as its target is always the optimal solutions. Actually, we could only run CPLEX for the five smaller cases due to its long running time. Both *Greedy* and *Iterative Greedy* are 2-4 orders of magnitude faster than CPLEX. It takes them about 200 seconds to solve the selection of 30 videos in a quality space with 500 points. From Section 6, we know the running time increases linearly with the number of media objects V. Thus, it may take a few hours to select replicas in a real media system with thousands of media objects. Fortunately, we do not need to run these algorithms very often and the running time of our online algorithm is very small, as we will see in Section 8.3.

8.2.1 Effects of Utility Functions

We test our algorithms with four types of utility functions: *hard-quality*, *financial*, *Manhattan distance*, and *minimum penalty*. They are ordered by the speed of utility loss as a function of distance in the quality space. The details of these utility functions can be found in our technical report [30]. We used Manhattan distance in the experiments presented in Fig. 11 and Fig. 12.

Fig. 13 shows the frequency of quality points chosen by *Greedy* in a 20×20 space for a total of 30 videos. In Fig. 13, any point (x, y, z) shows that, out of the 30 media objects,

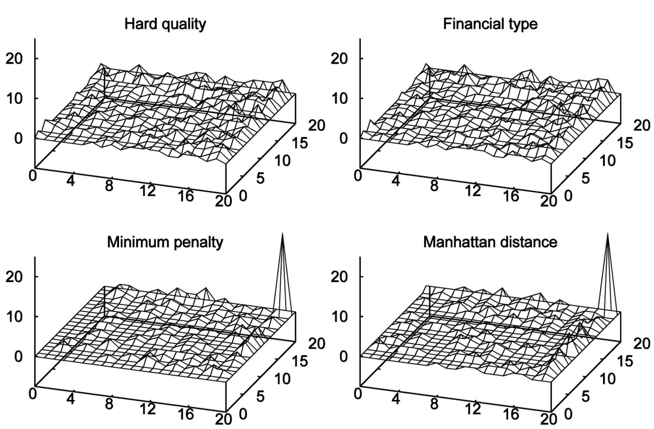

Fig. 13. Frequency of replicas chosen by *Greedy* in a 20×20 quality space.

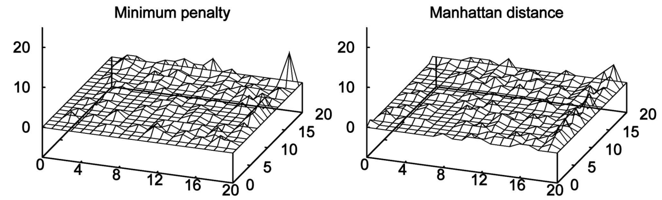

Fig. 14. Frequency of replicas chosen by *Iterative Greedy* in a 20×20 quality space.

z objects have replicas of quality (x, y) selected by the algorithm. Larger numbers on the X, Y axes mean lower quality. We can see that utility functions significantly affect the choice of replicas. For *hard-quality* and *financial*, whose utility drops very fast, the replicas are evenly distributed in the quality space. For the other two utility functions, *Greedy* selects more replicas with lower quality. A salient problem is that, for more than 20 videos, *Greedy* picks the lowest quality replica (19, 19). This confirms our discussion in Section 6.3: With overestimated utility rates, smaller replicas are always chosen first. The situation is improved by the *Iterative Greedy* algorithm. Fig. 14 shows the distribution of replicas after running *Iterative Greedy* with the same set of inputs. The high peaks on points (19, 19) disappear and the total utility rate increases by about 2 percent.

It should be noted that the solutions found by *Greedy* are almost optimal if we use *hard-quality* and *financial* types of utility functions. *Iterative Greedy* has no advantages under this situation. Our explanation to this is: By utilizing fast utility-dropping functions, we are making the FSRS problem a lot easier to solve. Recall (Section 6) that the major difficulty of solving FSRS comes from the combinatorial effects among replicas in collecting utility. However, the above utility functions tend to make replicas more isolated as they can only collect utility locally.

8.3 Dynamic and Distributed Replication

We test our dynamic replication algorithm for the soft-quality model for its optimality and speed. We simulate a system for a period of time during which events of query rate changes of media objects are randomly generated. We allow the query rate of videos to increase up to 20 times and to decrease down to 1/10 of the original rate. We first compare the total utility rate of the selected replicas between the online phase of SOFTDYNAREP and *Greedy*. In all cases, the replicas selected exactly match with those found by the M-GREEDY algorithm discussed in Section 7.3; thus, the utility rates are always the same between two solutions. As shown in Fig. 15a, the replicas selected by SOFTDYNAREP have utility rates that are consistently within 99.5 percent of that by the original *Greedy* algorithm. In this experiment with 270 videos and a 20×20 quality space, the running time of SOFTDYNAREP for each event is on the order of 10^{-4} seconds, while ADD-REPLICA needs to run about half a hour to solve the same problems. The main reason for SOFTDYNAREP's efficiency is the small number of storage exchanges. In Fig. 15b, we record such numbers for each execution of SOFTDYNAREP and very few of these readings exceeds 15.

We also study replica selection in a multiserver environment under the hard-quality model by comparing various

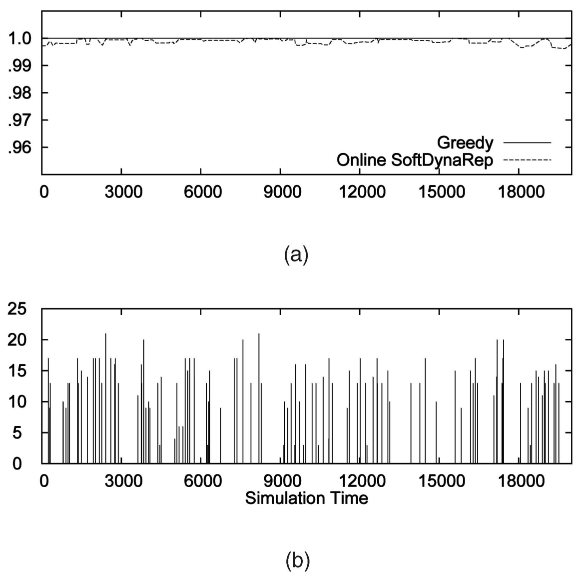

Fig. 15. Performance of SOFTDYNAREP. (a) Relative utility rate. (b) Number of exchanges.

load balancing strategies. Due to space limitations, readers are referred to [30] for more details.

9 CONCLUSIONS

In this paper, we studied the problem of selecting quality-specific replicas of media data. This problem is generally overlooked in multimedia database research due to the oversimplified assumption that storage space is abundant. We demonstrated by analysis and experiments that this is not the case if the system is to adapt to user quality requirements with reasonable granularity. We provided solutions to the problem under two different system models. In the discussions on a hard-quality system model, we concluded that the query rate and storage of individual replicas are the most critical factors that affect performance. We also proposed a greedy algorithm to solve the replica selection problem on a soft-quality system model. Experiments showed that the solutions found by our algorithm are within 3 percent of the optimal. An advanced version of this algorithm further reduced that to 1 percent. A derived online algorithm provided an elegant solution to an important subproblem of dynamic data replication.

APPENDIX A

REJECT (BLOCKING) PROBABILITY IN A GENERALIZED ERLANG MODEL

We use CPU as an example to elaborate this. The CPU requests from different replicas can be viewed as competitors for a shared resource pool with a finite capacity C. Kelly first studied the probability of rejection in such systems [31], [32]. The main idea is to analyze the occurrence of resource occupation states, denoted as $\vec{n} = (n_1, n_2, \cdots, n_M)$, where n_k is the number of requests to replica k currently being serviced. According to [32], the reject probability of any replica k is

$$P_k = \frac{\sum_{\vec{n}\in\mathcal{S}_k} \prod_{k=1}^{M} \frac{1}{n_k!} \lambda_k^{n_k}}{\sum_{\vec{n}\in\mathcal{S}} \prod_{k=1}^{M} \frac{1}{n_k!} \lambda_k^{n_k}}, \tag{16}$$

where $\mathcal{S}_k = \{\vec{n} : C - c_k < \sum_{k=1}^{M} n_k c_k \le C\}$ and $\mathcal{S} = \{\vec{n} : \sum_{k=1}^{M} n_k c_k \le C\}$ are two sets of states. The states in $\mathcal{S}_k$ are those at which a request to replica k will be rejected (as there are less than c_k units of resource available) while $\mathcal{S}$ is the collection of all possible states.

Due to the discrete feature of the states, it is very difficult to discuss the characteristics of (16). Fortunately, Gazdzicki et al. [33] give the following asymptotic approximation to (16):

Case 1. When the resource has a light load, i.e., $\sum_{k=1}^{M} \lambda_k c_k < C$, the class-specific reject probability is

$$P_k = e^{\tau d\epsilon - I(C)} \frac{d}{\sqrt{2\pi N}\sigma} \left(\frac{1 - e^{\tau c_k}}{1 - e^{\tau d}}\right)(1 + o(1)). \tag{17}$$

Case 2. When the resource has a critical load, i.e., $\sum_{k=1}^{M} \lambda_k c_k = C$, P_k becomes

$$P_k = \sqrt{\frac{2}{\pi N}} \frac{c_k}{\sigma}(1 + o(1)). \tag{18}$$

Case 3. When the pool is heavily loaded, i.e., $\sum_{k=1}^{M} \lambda_k c_k > C$, we get

$$P_k = (1 - e^{\tau c_k})(1 + o(1)), \tag{19}$$

where N is the *scale* of the resource pool (i.e., $N = \Theta(C)$), τ is the unique solution to the equation

$$\sum_{k=1}^{M} \frac{f_k}{\mu_k} c_k e^{\tau c_k} = C, \tag{20}$$

and other relevant quantities are defined as follows:

1. d is the greatest common divisor of $c_1, c_2, \dots, c_M$,
2. $\epsilon = \frac{C}{d} - \left[\frac{C}{d}\right]$, where $[a]$ denotes the largest integer such that $[a] \le a$,
3. $I(C) = \tau C - \sum_{k=1}^{M} \lambda_k (e^{\tau c_k} - 1)$, and
4. $\sigma^2 = \sum_{k=1}^{M} \lambda_k c_k^2 e^{\tau c_k}$.

APPENDIX B

PROOF OF PROPOSITION 1

Proof. The reject probability for group $\mathcal{A}$ is $P_A = \frac{1}{f_A} \sum_{i\in A} f_i P_i$, where P_i is the reject probability for traffic class i and $f_A = \sum_{i\in A} f_i$. Similarly, we have $P_B = \frac{1}{f_B} \sum_{i\in B} f_j P_j$. For overloaded resources, we can use (19) to quantify the quality-specific reject probability. Therefore, we get $P_i = 1 - e^{\tau_A c_i}$ and $P_j = 1 - e^{\tau_B b_j}$, where τ_A and τ_B are constants that satisfy (20). Let $s = \tau_A$ and $u = \tau_B$ and we call s and u the *passage coefficients*[12] of groups $\mathcal{A}$ and $\mathcal{B}$. To prove $P_A > P_B$, it is sufficient to show that

$$\frac{1}{f_A} \sum_{i\in A} f_i s^{c_i} < \frac{1}{f_B} \sum_{j\in B} f_j u^{b_j}.$$

We first apply a proportional scaling to the classes in group $\mathcal{A}$, that is, we increase all c_i as well as the total

12. These are basically the probabilities of one single unit of resource being free.

resource units R_A by a factor of ω $(\omega > 1)$ such that $\omega R_A = R_B$. According to [33], such "scaling" will not increase the class-specific reject probability, i.e., $\forall i$, $P_i' \leq P_i$, where $P_i' = 1 - s'^{\omega c_i}$ is the reject probability of class i after it is scaled. Note that s is replaced by a new constant s'. With this transformation, this proof is concluded if we can show

$$\frac{1}{f_A}\sum_{i\in A} f_i s'^{\omega c_i} < \frac{1}{f_B}\sum_{j\in B} f_j u^{b_j}. \quad (21)$$

Kelly [31] states that the passage coefficient of a traffic group can be approximated by the inverse of its load coefficient. Thus, we get $s' \approx \frac{1}{m_A}$ and $u \approx \frac{1}{m_B}$. Note that scaling does not change the load coefficient of group $\mathcal{A}$. For $m_A \gg m_B$, we have $s' < u$ for sure. With the given condition $\min_{i\in A}\{c_i\}/R_A > \max_{j\in B}\{b_j\}/R_B$, we immediately have $\min_{i\in A}\{\omega c_i\} > \max_{j\in B}\{b_j\}$, which further leads to $s'^{\omega c_i} < u^{b_j}$, $\forall i, j$. Having this, (21) is trivially correct. □

Appendix C
Proof of Theorem 1

Proof. Using the notations for the bandwidth resource (λ_k, b_k, B), we first derive an upper bound of P under the critical load situation. Recalling the asymptotic approximation to P_k for a critically loaded resource in (18), we immediately have $\tau = 0$ and $e^{\tau b_k} = 1$.

From (18), we also get $\lambda_k P_k^2 = \frac{2}{\pi N}\frac{\lambda_k b_k^2}{\sum_{k=1}^{M}\lambda_k b_k^2}$, which leads to

$$\sum_{k=1}^{M}\lambda_k P_k^2 = \sum_{k=1}^{M} f_k \mu_k^{-1} P_k^2 = \frac{2}{\pi N}. \quad (22)$$

To get the upper bound for $P = \frac{1}{f}\sum f_k P_k$, we use the method of Lagrangian multipliers with the following optimization function:

$$L = \sum f_k P_k - \phi\left(\sum f_k \mu_k^{-1} P_k^2 - \frac{2}{\pi N}\right),$$

where ϕ is the Lagrangian multiplier. We discuss how P_k may affect the bound of P given all f_k and μ_k. The condition for maximality is thus $\frac{\partial L}{\partial P_k} = 0$, $\forall k$. This is the same as

$$f_k - 2\phi\frac{f_k}{\mu_k}P_k = 0, \forall k.$$

Immediately, we get $P_k = \frac{\mu_k}{2\phi}$ as the condition for achieving the upper bound. Plugging this into (22), we have

$$2\phi = \sqrt{\frac{\pi N \sum f_k \mu_k}{2}}.$$

Let $\beta = \sum f_k \mu_k$, we have $P_k = \mu_k\sqrt{\frac{2}{\pi N \beta}}$ under the optimal situation. Therefore, the maximum value of P can be expressed as

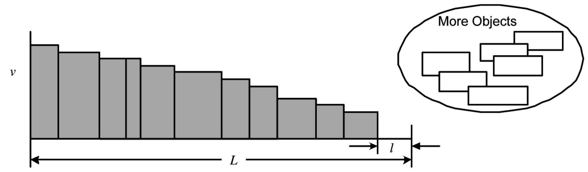

Fig. 16. A knapsack filled with objects.

$$P = \frac{1}{f}\sum f_k P_k = \frac{1}{f}\sum f_k \mu_k \sqrt{\frac{2}{\pi N \beta}} = \sqrt{\frac{2\beta}{\pi N f^2}}. \quad (23)$$

Now, we consider the underload situation (i.e., Case 1 in Appendix A). Note that any such case can be transformed into a critical load case by adding a new class (i.e., class $M+1$) of requests. Specifically, we let $b_{M+1} = B$ and choose an arbitrary λ_{M+1} such that $\sum_{k=1}^{M}\lambda_k b_k + \lambda_{M+1} b_{M+1} = B$. We now show that the reject probability P always increases after the transformation.

According to [34], P_k can be obtained from the following relation:

$$\sum_{k=1}^{M+1}\lambda_k b_k q(j - b_k) = jq(j), \qquad j = 0, 1, \cdots, B, \quad (24)$$

where $q(j)$ is the stationary probability that exactly j units of resources are occupied and $q(j) = 0$ for $j < 0$. It is easy to see that $\sum_{j=0}^{B} q(j) = 1$ and P_k is given by $P_k = \sum_{i=0}^{b_k - 1} q(B - i)$.

Running (24) recursively with the unknown quantity $q(0)$ as the base case, we have

$$q(0) + q(1) + \cdots + q(B) = q(0)(1 + \alpha_1 + \cdots + \alpha_B) = 1, \quad (25)$$

where α_j $(1 \leq j \leq B)$ is a constant determined by the recursions. By adding class $M+1$ and reconsidering (24), (25) becomes

$$\begin{aligned} & q(0) + q(1) + \cdots + q(B) \\ = \; & q(0)[1 + \alpha_1 + \cdots + (\alpha_B + \lambda_{M+1} b_{M+1})] = 1. \end{aligned} \quad (26)$$

As a result, the value of $P_k = \sum_{i=0}^{b_k - 1} q(B - i)$ for any class k is larger in (26) than in (25). Therefore, quantity P in the underload case is smaller than the corresponding critical load case generated by the above transformation. In other words, it is also bounded by $\sqrt{\frac{2\beta}{\pi N f^2}}$. □

Appendix D
Optimality of a Simple Solution to the 0/1 Knapsack Problem

In the 0-1 Knapsack problem, each candidate object has its own size and value. We define the ratio of value over size as the value density of an object (denoted as v). We claim that if we put the objects with the largest value density into the knapsack, the total value obtained are near-optimal when the size of the knapsack is far greater than the size of any individual object.

As illustrated in Fig. 16, the knapsack has size L and the Y axis represents value density. Each candidate object is represented as a rectangle and its size as the width, and value as the area of the rectangle, respectively. Our algorithm will fill the knapsack with objects with the largest value densities until no objects can be filled as a whole. It is easy to see that, if all L storage is filled, the solution is optimal as any other plan will decrease the total value achieved. If there is a unfilled region with size l, we can fill it with the object whose value density v' is the largest among the unselected objects. This generates an infeasible solution as we have to cut a piece (with size l) from the last object. However, it gives an upper bound of the optimal total value: $\hat{v} = \Omega + lv'$, where Ω is the achieved total value (area of the shaded region in Fig. 16). Here, lv' can be viewed as an upper bound of the difference between our solution and the optimal value. When $L \gg l$, we have $lv' \ll \hat{v}$.

Acknowledgments

The authors would like to thank Professor Leming Qu, Professor Xingquan Zhu, and Mr. Shan Lei for sharing with them their valuable insights in the early stages of this study. They are also grateful to Professor Hong Wan for her help with the CPLEX optimization software. This work was supported by US National Science Foundation grants IIS-0534702 and IIS-0242421.

References

[1] Y.-C. Tu, S. Prabhakar, A. Elmagarmid, and R. Sion, "QuaSAQ: An Approach to Enabling End-to-End QoS for Multimedia Databases," *Proc. Int'l Conf. Extending Database Technology (EDBT '04),* pp. 694-711, Mar. 2004.

[2] S. Nepal and U. Srinivasan, "DAVE: A System for Quality Driven Adaptive Video Delivery," *Proc. Int'l Workshop of Multimedia Information Retrieval (MIR '04),* pp. 224-230, 2004.

[3] A. HAfiD and G. Bochmann, "An Approach to Quality of Service Management in Distributed Multimedia Application: Design and Implementation," *Multimedia Tools and Applications,* vol. 9, no. 2, pp. 167-191, 1999.

[4] B. Noble, M. Price, and M. Satyanarayanan, "A Programming Interface for Application-Aware Adaptation in Mobile Computing," *Proc. Second USENIX Symp. Mobile Computing,* Apr. 1995.

[5] R. Mohan, J.R. Smith, and C.-S. Li, "Adapting Multimedia Internet Content for Universal Access," *IEEE Trans. Multimedia,* vol. 1, no. 1, pp. 104-114, 1999.

[6] E. Amir, S. McCanne, and H. Zhang, "An Application Level Video Gateway," *Proc. ACM Multimedia,* pp. 255-265, 1995.

[7] M. Nicola and M. Jarke, "Performance Modeling of Distributed and Replicated Databases," *IEEE Trans. Knowledge and Data Eng.,* vol. 12, no. 4, pp. 645-672, July/Aug. 2000.

[8] C.-F. Chou, L. Golubchik, and J.C.S. Lui, "Striping Doesn't Scale: How to Achieve Scalability for Continuous Media Servers with Replication," *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS '00),* pp. 64-71, Apr. 2000.

[9] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen, "A Scalable Solution to the Multi-Resource QoS Problem," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1999.

[10] Y.-C. Tu, J. Yan, and S. Prabhakar, "Quality-Aware Replication of Multimedia Data," *Proc. Database and Expert Systems Applications (DEXA '05),* pp. 240-249, Aug. 2005.

[11] E. Bertino, A. Elmagarmid, and M.-S. Hacid, "A Database Approach to Quality of Service Specification in Video Databases," *SIGMOD Record,* vol. 32, no. 1, pp. 35-40, 2003.

[12] J. Walpole, C. Krasic, L. Liu, D. Maier, C. Pu, D. McNamee, and D. Steere, "Quality of Service Semantics for Multimedia Database Systems," *Proc. Data Semantics 8: Semantic Issues in Multimedia Systems,* vol. 138, 1998.

[13] G. On, J. Schmitt, M. Liepert, and R. Steinmetz, "Replication with QoS Support for a Distributed Multimedia System," *Proc. 27th EUROMICRO Conf.,* Sept. 2001.

[14] M. Rabinovich, "Issues in Web Content Replication," *Data Eng. Bull.,* vol. 21, no. 4, pp. 21-29, 1998.

[15] L. Qiu, V.N. Padmanabhan, and G.M. Voelker, "On the Placement of Web Server Replicas," *Proc. IEEE INFOCOM,* pp. 1587-1596, 2001.

[16] A. Milo and O. Wolfson, "Placement of Replicated Items in Distributed Databases," *Proc. Int'l Conf. Extending Database Technology (EDBT '88),* pp. 414-427, 1988.

[17] T.D.C. Little and D. Venkatesh, "Popularity-Based Assignment of Movies to Storage Devices in a Video-on-Demand System," *Springer/ACM Multimedia Systems,* vol. 2, no. 6, pp. 280-287, Jan. 1995.

[18] Y. Wang, J.C.L. Liu, D.H.C. Du, and J. Hsieh, "Efficient Video File Allocation Schemes for Video-on-Demand Services," *Springer/ACM Multimedia Systems,* vol. 5, no. 5, pp. 282-296, Sept. 1997.

[19] A. Dan and D. Sitaram, "An Online Video Placement Policy Based on Bandwidth to Space (BSR)," *Proc. ACM SIGMOD,* pp. 376-385, 1995.

[20] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu, "Approximation Algorithms for Data Placement on Parallel Disks," *Proc. Symp. Discrete Algorithms (SODA '98),* pp. 223-232, 1998.

[21] O. Wolfson, S. Jajodia, and Y. Huang, "An Adaptive Data Replication Algorithm," *ACM Trans. Database Systems,* vol. 22, no. 4, pp. 255-314, June 1997.

[22] D. Kossman, M. Franklin, and G. Drasch, "Cache Investment: Integrating Query Optimization and Distributed Data Placement," *ACM Trans. Database Systems,* vol. 25, no. 4, pp. 517-558, 2000.

[23] 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,* vol. 11, pp. 35-62, 2000.

[24] W.Y. Lum and F.C.M. Lau, "On Balancing between Transcoding Overhead and Spatial Consumption in Content Adaptation," *Proc. MOBICOM,* pp. 239-250, Sept. 2002.

[25] V. Harinarayan, A. Rajaraman, and J.D. Ullman, "Implementing Data Cubes Efficiently," *Proc. ACM SIGMOD,* pp. 205-216, June 1996.

[26] R.B. Cooper, *Introduction to Queueing Theory.* North Holland, 1981.

[27] G. Menges, *Economic Decision Making: Basic Concepts and Models,* chapter 2, pp. 21-48. Longman, 1973.

[28] Z. Drezner and H.W. Hamacher, *Facility Location: Applications and Theory.* Springer, 2002.

[29] O. Kariv and S.L. Hakimi, "An Algorithmic Approach to Network Location Problems. II: The p-Medians," *SIAM J. Applied Math.,* vol. 37, no. 3, pp. 539-560, 1979.

[30] Y.-C. Tu, J. Yan, G. Shen, and S. Prabhakar, "Multi-Quality Data Replication in Multimedia Databases," Technical Report CSD-TR-06-009, Purdue Univ., 2006.

[31] F.P. Kelly, "Blocking Probabilities in Large Circuit-Switched Networks," *Advances in Applied Probability,* vol. 18, no. 2, pp. 473-505, June 1986.

[32] F.P. Kelly, "Loss Networks," *Annals of Applied Probability,* vol. 1, no. 3, pp. 319-378, Aug. 1991.

[33] P. Gazdzicki, I. Lambadaris, and R. Mazumdar, "Blocking Probabilities for Large Multirate Erlang Loss Systems," *Advances in Applied Probability,* vol. 25, pp. 997-1009, Dec. 1993.

[34] J.S. Kaufman, "Blocking in a Shared Resource Environment," *IEEE Trans. Comm.,* vol. 29, no. 10, pp. 1474-1481, Oct. 1981.

Yi-Cheng Tu received the bachelor's degree in horticulture from Beijing Agricultural University (Beijing, China), the MS degree in computer science in 2003 from Purdue University, and is currently a PhD student in the Department of Computer Sciences at the same university. His current research efforts address load management in data stream management systems for the purpose of meeting real-time requirements of user queries, and application of control theory in self-tuning database systems. He has also worked on performance analysis of peer-to-peer systems, QoS-aware query processing, and data placement in multimedia databases. He is a student member of the IEEE and ACM.

Jingfeng Yan received the BS degree in chemistry from Eastern China University of Science and Technology (Shanghai, China) in 1996 and the master's degrees in chemistry and computer science from Peking University (Beijing, China) in 1999 and Depaul University (Chicago, Illnois) in 2002, respectively. He is a PhD student in the Department of Computer Sciences at Purdue University. His current research is on query optimization in streaming database. He has also worked on performance evaluation and tuning for video database management system. He is a student member of the IEEE.

Gang Shen received the BA degree in economics from Fudan University, China, in 1992, the MS degree in applied statistics from Worcester Polytechnic Insitute (WPI) in 2004, and is currently a PhD student in the Department of Statistics at Purdue University. Prior to joining the MS program at WPI, he hold positions in DHL (as a project manager) and China Merchants Bank (as a credit standing analyst). His research interests lie in the fields of stochastic processes, time series analysis, and Bayesian statistics. He is a member of American Statistics Association (ASA).

Sunil Prabhakar received the BTech degree in electrical engineering from the Indian Institute of Technology, Delhi, in 1990 and the MS and PhD degrees in computer science from the University of California, Santa Barbara in 1998. He is an associate professor of computer sciences at Purdue University. He has served on the editorial boards of the *Distributed and Parallel Databases* journal, the *Journal of Database Management*, and the *Journal of Ubiquitous Computing and Intelligence*. He has served on the program committees of leading conferences including SIGMOD, VLDB, and ICDE. He is the recipient of the US National Science Foundation CAREER award. He is a senior member of the IEEE.