Quality-Aware Replication of Multimedia Data

Yi-Cheng Tu, Jingfeng Yan, and Sunil Prabhakar
Department of Computer Sciences, Purdue University
West Lafayette, IN 47906, USA
{tuyc, sunil}@cs.purdue.edu

ABSTRACT

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. Users can request data with a specific quality requirement 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 pre-compute 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. Although replication has been well studied, to the best of our knowledge, the problem of multiple-quality replication has not been addressed. 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. The hard quality problem is reduced to a 0-1 Knapsack problem and we propose an efficient solution that minimizes the probability of request rejection due to unavailability of the requested quality replica. For the soft quality system, an important optimization goal is to minimize utility loss. We propose a powerful greedy algorithm to solve this problem. Extensive simulations show that our algorithm performs significantly better than other heuristics. The algorithm is flexible in that it can be extended to deal with problems of distributed data replication and changes of query pattern.

Categories and Subject Descriptors

H.1 [**Information Systems**]: Models and Principles

Technical Report CSD-TR-0423, Department of Computer Sciences, Purdue University

General Terms

performance, algorithms

Keywords

quality adaptation, integer programming, data replication

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 [24, 19, 9] 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 [18, 19]. The quality parameters of interest also differ by the type of media we deal with. For digital video, the quality parameters of interest include resolution, frame rate, color depth, signal-to-noise ratio (SNR), audio quality, compression format, and security level [24]. 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: i) store only the highest resolution copy, and convert it to the quality format requested by the user as needed at run-time; or ii) pre-compute each different quality that can be requested and store them on disk. When the user query is received, the appropriate copy is retrieved from 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 [18]. Therefore online transcoding is difficult in a multi-user environment. Our experiments (Fig 1) run on a 2.4GHz Pentium 4 CPU confirm this claim: a MPEG1 video is transcoded at a speed of only 15 to 60 frames per second. This corresponds to 0.6 to 2.4 times of the entire CPU power if the frame rate for the video is 25fps. We can see that CPU power is the bottleneck if we depend on online transcoding. As a result, many *transcode proxy* servers or video gateways [1] 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 multiple quality 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.

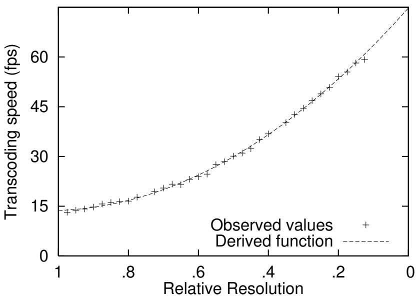


Figure 1: Time for transcoding a 640×480 MPEG1 video to various resolutions.

Static adaptation systems are designed under the assumption that either *there is always enough storage space* or *user requests concentrate on a small number of quality profiles.* However, this may not be true for a real-world multimedia database. First of all, users vary widely in their quality needs and resource availability [18]. This leads to a large number of quality-specific copies of the same media content that need to be stored on disk. Secondly, 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 3 shows the disk space needed to accommodate all possible quality profiles could be intolerably high. Therefore, the choice of which quality copies to store becomes important and is the focus of this paper.

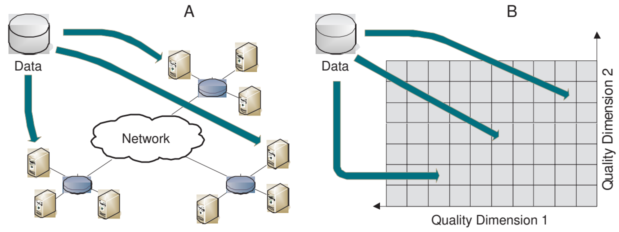


Figure 2: Traditional (A) and quality-aware (B) data replication.

We view the selection of media copies for storage as a data replication problem (Fig 2). Traditional data replication focuses on placement of copies of data in various nodes in a distributed environment [20]. Our quality-aware replication of multimedia deals with data placement in a metric space of quality values (termed as *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 multiple quality copies generated via transcoding. In this paper, we present strategies to choose quality of replicas under two different assumptions about user behaviors: *Hard Quality* and *Soft Quality*. Under the hard quality model, users must receive the exactly quality requested. If such a quality is not already stored on disk, it must be generated by transcoding from an available quality. If the resources necessary for this transcoding are not available (e.g. due to too many requests) 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: *reject rate* of requests, *user satisfaction*, and *resource consumption* [4, 13]. Our data replication algorithms are designed to achieve the lowest rejection rate, or highest user satisfaction under fixed resource (CPU, bandwidth, and storage) capacities. To the best of our knowledge, this is the first work to study quality-aware data replication. We hope our work will provide useful guidelines to system designers in building cost-effective and user-friendly multimedia databases. The major contributions of this paper are:

1. We analytically and experimentally show that the storage cost of static adaptation is so high that typically only a small number of replicas in the quality space can be accommodated in disks;
2. In a hard quality-aware system where users are assumed to be strict on quality requirements, we develop a (near-optimal) replica selection algorithm that minimizes request reject rate;
3. We formulate the replica selection under a soft quality model as a 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 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.

The remainder of this paper is organized as follows: we first introduce the system model in Section 2; Section 3 discusses storage use of the replication process; we present our replica selection algorithms in Sections 4, 5, and 6; Section 7 is dedicated to experimental results; related works are summarized in Section 8; we conclude the paper by Section 9.

2. SYSTEM MODEL 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. We list in Table 1 the notations that will be used throughout this paper.

In our model, a server is characterized by the total amount 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. In modern media servers, network bandwidth is most likely to be the bottleneck.

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 ($\vec{q} = \{q_1, q_2, ..., q_m\}$, termed as *quality vector*). Each quality vector can thus be modeled as a point (hereafter called *quality point*) in a m-dimensional

Table 1: **Notations and definitions.**

Symbol	Definition
System parameters	
N	Number of media servers
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, $f = \sum_{k=1}^{M} f_k$
Replica-specific parameters	
f_k	Query rate, number of requests per unit time
μ_k	Service rate, requests served per unit time
c_k	CPU cycles per unit time for transcoding into this quality point from original quality
b_k	Bandwidth needed for streaming
s_k	Storage space needed if a replica is placed

space. Generally, the domain of a quality parameter contains 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 1920×1080 (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 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, b_k. 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 $1/\mu_k$ may not be the same as the standard playback time of the media as the users may use VCR functionalities (e.g. stop, 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 precisely estimated from empirical functions derived by regression (see Section 3). Note c_k is fixed as the transcoding cost only depends on the target quality.

Upon receiving a request to a media, the server:

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 transcodes a copy from a high-quality replica (by consuming c_k units of CPU);
3) rejects the request if not enough CPU is available.

If either 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 sufficient bandwidth is unavailable. We ignore the CPU costs of non-transcoding operations as they are trivial compared to transcoding costs and do not change with the specified $\vec{q}$. In the above model, requests are either admitted or rejected. In Section 5, we study a more flexible model where users may compromise the original quality they specify.

2.1 Assumptions

In this paper, we assume that CPU is a heavily overloaded

Table 2: Total relative storage in a 3D space.

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

resource as a result of online transcoding requests. On the other hand, system bandwidth is moderately overloaded. Formally, we can say $\sum_{k=1}^{M} \frac{f_k c_k}{\mu_k} \gg C$ and $\sum_{k=1}^{M} \frac{f_k b_k}{\mu_k} = pB$ where p ($p \geq 1$) is a small number. This is reasonable due to our discussion in Section 1 about CPU being the bottleneck in our system model.

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.

3. STORAGE REQUIREMENTS FOR QUALITY-AWARE REPLICATION

As mentioned in Section 1, it is often assumed in previous works that enough storage is available for static adaptation. Now we explore 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 [19], the bitrate of a video replica with a single reduced quality parameter (e.g. resolution) is expressed as:

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

where F_0 is the bitrate of the original video, R is the percentage of quality change ($0 \leq R \leq 1$) from the original media, and β is a constant derived from experiments ($2 > \beta > 1$). 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. $R_i = \frac{i}{n}$). The sum of the bitrate of all copies is given by:

$$\begin{aligned}
\sum_{i=0}^{n} F_0(1 - R_i^{\frac{1}{\beta}}) &= F_0\left(n - \sum_{i=0}^{n}\left(\frac{i}{n}\right)^{\frac{1}{\beta}}\right) \\
&\approx F_0\left(n - \int_0^n \left(\frac{i}{n}\right)^{\frac{1}{\beta}}\right) \qquad (2) \\
&= F_0\left(n - \frac{n\beta}{\beta+1}\right) \\
&= F_0\frac{n}{\beta+1} = F_0 O(n). \qquad (3)
\end{aligned}$$

The corresponding storage requirement can be easily calculated as $TF_0\frac{n}{\beta+1}$ where T is the playback time of the media. Note that the above only considers one quality dimension. In [19], Equation (1) is also extended to three dimensions (spatial resolution, temporal resolution, and SNR):

$$F = \alpha F_0(1 - R_A^{\frac{1}{\beta}})(1 - R_B^{\frac{1}{\gamma}})(1 - R_C^{\frac{1}{\theta}}) \qquad (4)$$

where R_A, R_B, and R_C are quality change in the three dimensions, respectively. The constants of their transcoder(s) are: $\alpha = 1.12$, $\beta = 1.5$, $\gamma = 1.7$, and $\theta = 1.0$. Using the same technique of approximation by integration as used in Equation (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 original size) needed for static adaptation

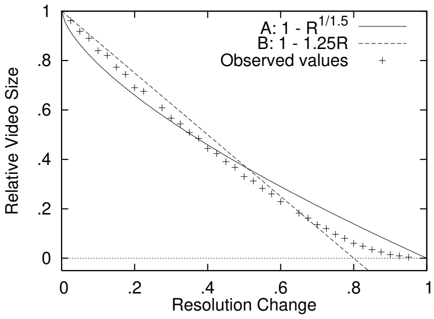


Figure 3: Change of video bandwidth with resolution degradation.

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. Some of the storage costs generated using Equation (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. We use the open-source video processing tool named *transcode* [1] in these experiments. Figure 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 Equation (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 Equation (3). The area of the triangle formed by X, Y axles and line B is $\frac{2}{5}$, which is the same as that given by Equation (3) since $\beta = \frac{3}{2}$. The fact that the areas under these three curves are very close to each other corroborates our analysis in this section.

4. 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 the users are not willing to negotiate when the quality he 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 an earlier section 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 non-availability of a requested quality results in the 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 $(r_1, r_2, ..., r_M)$ with 0/1 elements ($r_k = 1$ if replica k is to be stored in disk). Formally, the replica placement problem is to

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

where 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, by applying our replication algorithm, the M quality points are divided into two disjoint sets: a set A containing replicated points and a set B with non-replicated points. Following Equation (5), we have

$$P = \frac{1}{f}\big(f_A P_A + f_B P_B\big) \tag{6}$$

where $f_A = \sum_{i \in A} f_i$ is the total request rate in set A, $P_A = \frac{1}{f_A} \sum_{k \in A} f_k P_k$ is the reject probability of all requests from A and f_B, P_B are counterparts of f_A, P_A in set B.

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

$$P_A = P_A^{(b)} + P_A^{(c)} - P_A^{(bc)} \tag{7}$$

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

$$P_B = P_B^{(b)} + P_B^{(c)} - P_B^{(bc)} \tag{8}$$

where $P_B^{(b)}$, $P_B^{(c)}$, and $P_B^{(bc)}$ are defined according to requests from set B.

As no rejection by CPU will occur when a replica is placed in disk (Section 2.1), we have $P_A^{(c)} = 0$, which leads to $P_A^{(bc)} = 0$ and thus $P_A = P_A^{(b)}$. Plugging this and Equation (8) into (6), we have

$$P = \frac{1}{f}\left(f_A P_A^{(b)} + f_B \left(P_B^{(b)} + P_B^{(c)} - P_B^{(bc)} \right) \right) \tag{9}$$

We now establish the following theorem that will help analyze the above expression.

THEOREM 4.1. *For a group of M Poisson-arrival requests for replicas, if $\sum_{i=1}^{M} \frac{f_k c_k}{\mu_k} \gg C$, $P \approx 1$. All notations follow the same definitions in Table 1.*

PROOF. See Appendix A. □

In other words, Theorem 4.1 states that when the request load put on a resource is far greater than the its capacity, the reject probability is very close to 1. As discussed earlier in Section 2.1, the CPU in our quality-aware system is exactly such an overloaded resource. This gives $P_B^{(c)} \approx 1$ and thus $P_B^{(bc)} \approx P_B^{(b)}$. Revisiting Equation (9), we have

$$P \approx \frac{1}{f}\big(f_A P_A^{(b)} + f_B P_B^{(c)}\big) \approx \frac{1}{f}\big(f_A P_A^{(b)} + f_B\big). \tag{10}$$

Now we can see that, in order to get the smallest possible P, we can maximize f_A. In other words, we can minimize f_B as

[1] http://www.theorie.physik.uni-goettingen.de/∼streich/transcode/

$f_A + f_B = f$. By putting a replica k in set B, we have a net increase of f_k/f for P. But if it is put into A, the increase of P will be discounted by $P_A^{(b)}$. Thus, such a solution is effective as long as $P_A^{(b)} < P_B^{(c)} \approx 1$. This condition holds true in our system: since bandwidth is critically loaded or slightly overloaded (Section 2.1), the reject probability on bandwidth is significantly smaller than 1[2].

The above analysis is nice in that it shows the selection algorithm does not need to consider μ_k, c_k, and b_k even though they play a role in determining $P^{(b)}$ and $P^{(c)}$ (Equation 19, Appendix A). Now the problem becomes: how to get as large (small) a f_A (f_B) value as possible given the storage constraint S. This is obviously a 0-1 Knapsack problem. A good heuristic is as follows: sort all possible replicas by their request rate per unit size (f_k/s_k) and select those with the highest such values till the total storage is filled – a $O(M \log M)$ algorithm. In Appendix C, we prove 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.

5. SOFT QUALITY SYSTEMS

In hard quality systems, replicas of the same media object 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 not served at all. However, human users can tolerate some changes of quality [19] and the quality parameters specified by a user only represent the most desirable situation he wants. If these parameters cannot be exactly matched by the server, they are willing to accept another 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.

5.1 Utility Functions

We generally use *utility* to quantify user satisfaction on a service received [16]. 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 [13]. We thus set the goal of our replica selection strategies to be maximizing utility. The server operations shown in Section 2 needs to be modified in soft quality systems. For simplicity, we assume the 'renegotiation' process between 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.

Figure 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 [16]. This makes our replica selection a trivial problem: always keep the one with the highest quality. However, in a more realistic environment, the cost of the extra quality may be high as more resource have to be consumed (Section 1) on the client side. Thus excessively high

[2]This is intuitively obvious yet a rigorous mathematical justification [7] is non-trivial. See Appendix B for more details.

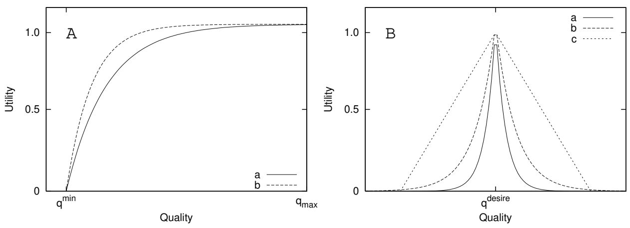


Figure 4: Different types of utility functions.

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 the functions do not have to be symmetric on both sides of q^{desire}. The hard quality model in Section 4 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 dimensional utility described above [13].

5.2 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 5.5.

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

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

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 5.1). 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 can not exceed S. We name our problem the *fixed-storage replica selection* (FSRS) problem and it can be formulated as the following integer programming:

$$\begin{aligned}
\text{maximize} \quad & \textstyle\sum_{j \in J} \sum_{k \in J} f_j u(j,k) Y_{jk} && (11)\\
\text{subject to} \quad & \textstyle\sum_{k \in J} X_k s_k \le S, && (12)\\
& \textstyle\sum_{k \in J} Y_{jk} = 1, && (13)\\
& Y_{jk} \le X_k, && (14)\\
& Y_{jk} \in \{0, 1\}, && (15)\\
& X_k \in \{0, 1\} && (16)
\end{aligned}$$

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 (12) shows the storage constraint while Equations (13) and (14) mean all requests from k should be served by one and only one replica. Here f_j, s_k, and S are inputs and X_k for all $k \in J$ is the solution.

The FSRS problem looks similar to a group of well-studied optimizations known as the *uncapacitated facility location* (UFL) problems [6]. Yet it is different from any known UFL

```
Algorithm GREEDY
Inputs: f_k, s_k for all replicas, total storage S
Output: a set of selected replicas, list
1  storage ← S, slist ← ∅, k ← 0
2  while k ≠ −1 do
3      k ← ADD-REPLICA(storage, list)
4      storage ← storage − s_k
5      append k to slist
6  return list

ADD-REPLICA (s, slist)
1  i ← −1, V_max ← 0
2  for each point k in the quality space do
3      if k ∉ slist and s_k ≤ s
4         U ← 0
5         for each point j in the quality space
6            U ← U + MAXUTILITY(j, k, slist)
7         if U/s_k > V_max
8            V_max ← U/s_k
9            i ← k
10 return i
```

Figure 5: The *Greedy* algorithm.

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 Equation (12) becomes $\sum X_k = p$, meaning only p ($p < |J|$) facilities are to be built. As the p-median problem is NP-hard [10], we can thus conclude FSRS is also NP-hard.

THEOREM 5.1. *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. □

5.3 The *Greedy* Algorithm.

Like in the Knapsack problem, we can use a benefit/cost model to analyze the quality of a replica k: the cost is obviously the storage s_k, the benefit would be the gain of utility rate of selecting k . What makes the problem more complicated is that the benefit is not fixed: it depends on the selection of other replicas. More specifically, the value of point k is the total utility rate of the set of points it serves and different selections of other replicas will affect the membership of this set of points. 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. It should be noted that this is different from choosing the replicas simply in descending order of precomputed $\Delta U_k/s_k$ because the ΔU_k changes depending upon which replicas have already been selected and thus must be recomputed after each selection.

Figure 5 shows the pseudo-code 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 $O\big(IM_i^2\big)$ for one media i. Here I is the number of replicas eventually placed in *list*. In the worst case when all points are selected, $I = M_i$. In our storage constrained system, I should be asymptotically smaller than M_i.

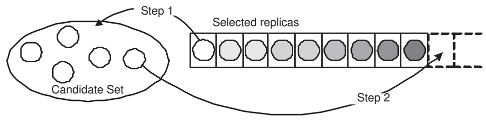


Figure 6: Replica replacement in Iterative Greedy.

Effects of the type of utility functions. It is easy to see that the shape of the utility functions affect 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 rate of all candidates have little difference at that moment. In Section 5.4, 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, 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.

5.4 The *Iterative Greedy* Algorithm

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 (existence of other selected replicas) is leveraged in its selection. In this sense, the first replica is the most 'unreliable' one: it is chosen taking no such global information into account. 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, as illustrated in Fig 6. Note that the one that is eliminated is also a candidate of the selection process.

The operations in *Iterative Greedy* are shown in Fig 7. All replicas selected by *Greedy* are stored in a FIFO queue

```
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
```

Figure 7: The *Iterative greedy* algorithm.

slist. In each iteration, we dequeue *slist* and find one replica (line 6) based on replicas remaining. 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 a fraction of that in *Greedy*.

5.5 Handling Multiple Media Objects

With very few modifications, both *Greedy* and *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: 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 $O(IV\hat{M}^2)$.

Note some quality parameters for physical media objects also lack adaptability. Video format is a good example. Without degradation of bitrate, replication along these quality dimensions requires even more storage than adaptable dimensions. However, it can reduce the time complexity of GREEDY the same way as the *content* dimension does.

Another point is that we set a constraint of replicating at least one copy for the video in Equation (13). In the multi-object scenario, we can further relax or tighten this constraint. To relax it, we allow zero replica being selected for a video, modifying Equation (13) 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 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 multi-video environment with the relaxed constraint.

6. EXTENSIONS

In this section we address the problem of multiple distributed servers instead of the single centralized server considered thus far, and the problem of changes in the access pattern for the various objects.

6.1 Distributed Data Replication

In Section 4 and 5 we discuss 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 the N servers[3]. As we can see, the analysis we show in Section 3 still holds true and we can use Equation (10) to guide our replica selection: the strategy is again to maximize f_A. When we obtain a set of replicas with the largest possible f_A, how to assign these replicas to N servers becomes a problem. Different methods may derive different $P_A^{(b)}$ values and we want to minimize it. 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 can be found in Appendix D.

We prefer a load balancing approach based on the idea of *resource pricing* proposed in [2]. In this approach, we set a *price* for the resource on which load is placed in each server. The price is set to reflect the supply-demand relationship of the resource. In our problem, the price for bandwidth can be set to

$$\psi_{bandwidth} = N^{B'/B} \tag{17}$$

where B' is the load put on bandwidth so far. The replica placement is accomplished by putting replicas one by one into a server with the lowest cost. Note in our problem we need to balance both storage and bandwidth. Therefore, the cost of placing replica k in a server is:

$$Cost = s_k \psi_{storage} + \frac{f_k}{\mu_k} b_k \psi_{bandwidth}. \tag{18}$$

Resource prices are updated upon placement of each replica according to Equation (17). The advantages of this algorithm are: server capacities do not have to be identical and it is proved to be $O(\log N)$-competitive [2].

Same strategy can be deployed to balance load under the soft quality system model even though reject probability is not the primary optimization goal.

6.2 Dynamic Data Replication

In previous sections we deal with the situation of *static* data replication, in which access rates of all qualities do not

[3] If each server only handles requests from its local region, the problem is not interesting as we only need to perform single-server replication at each server.

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 [15]; 2. Conclusions drawn from static replication studies form the basis of dynamic replication research [14].

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 terms of result optimality.

6.2.1 Hard quality systems

Our replication strategy for hard quality systems is easily adaptable to dynamic situations: 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 algorithm is displayed in Fig 8. Recall from Section 4, all replicas belong to either the replicated set A or the non-replicated set B. In HARDDYNAREP, we set a bound $\bar{\eta}$ such that for any replica k, we have $\eta_k > \bar{\eta} \Leftrightarrow k \in A$ and $\eta_k < \bar{\eta} \Leftrightarrow k \in B$. HARDDYNAREP is called when we detect a change of access rate for a replica r. Replication decision is made based on comparison between the new η_r and the bound $\bar{\eta}$. The time complexity of this algorithm is $O\big(\log M\big)$.

```
Algorithm HARDDYNAREP
Inputs: sorted list L of all replicas, η̄, and a replica r
  reinsert η_r into L
  Case 1. f_r increases
    Case 1.1. r was replicated, do nothing
    Case 1.2. r was not replicated
              if η_r > η̄
              do reset bound η̄
                 ∀k, if k ∈ A and η_k < η̄, dereplicate k
                 replicate r
  Case 2. f_r decreases, operations are opposite to Case 1.
```

Figure 8: Hard quality dynamic replication algorithm.

6.2.2 Soft quality systems

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 some nice features that we can 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 [15, 26], we understand a solution for more general situation is meaningful and we leave it as future work.

Let us first investigate how ADD-REPLICA, being the core of both *Greedy* and *Iterative Greedy*, selects replicas. The

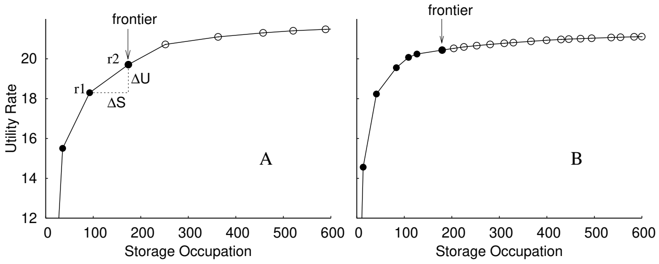


Figure 9: Replication roadmap.

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 9 shows two examples of RRs plotted with the same scale. First of all, we have the following feature of RR:

LEMMA 1. *A Replication Roadmap is convex.*

PROOF. In a RR, the slope of the line connecting any two consecutive points (e.g. $r1$ and $r2$ in Fig 9) 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 non-increasing. □

We can also draw RRs for individual media objects. For the same reason as in Lemma 2, 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 9 as an example. Suppose we have fully extended RRs: all future replicas are precomputed (empty dots in Fig 9) 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 backwards 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 following online algorithm for dynamic replication:

The algorithm consists of two phases: the *Preprocess Phase* and *Online Phase*. In the first phase, we need to 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 $O\big(V\hat{M}^3\big)$ 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 $O\big(I \log V\big)$ where I is the number of storage exchanges (line 9). In the worst case where most of

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

```
Algorithm SoftDynaRep
Preprocess Phase
  run Greedy or Iterative Greedy
  for RRs of all V media objects
      store the post-frontier segment in flist
      store the pre-frontier segment in blist
      extend RR
Online Phase
  Case 1. f_i increases
  recalculate slopes of stored segments for i
  update blist and flist (reinsertion)
  while there is room to improve total U
  do take storage from media j at the tail of blist
      update frontiers of both i and j
      insert pre/post-frontier segments of i and j into blist and flist
  Case 2. f_i decreases, symmetric to Case 1
```

Figure 10: Soft quality dynamic replication algorithm.

i's replicas are to be stored, we have $I = O(M_i)$. The case of query rate decrease is just handled in an opposite way to what we have discussed above.

Optimality of SoftDynaRep**.** We claim that the online phase of SoftDynaRep achieves the same quality in the selected replicas as that by rerunning *Greedy*. A proof can be found in Appendix E. In summary, *Greedy* essentially recreates a list of all $V\hat{M}$ replicas (the system RR) and selects the ones with the highest utility density values to store. In SoftDynaRep, we break the list into V small lists (media-specific RR) and achieve the same utility density ordering by dealing with only the head (frontier) of each list.

7. EXPERIMENTS

We study the behavior of various algorithms described in previous sections by extensive simulations. We use traces of 270 MPEG1 videos extracted from a real video database as experimental data. For all replicas, we set their μ_k to be their standard playback time. Some of the videos are then transcoded into replicas of different spatial resolution and frame rates using *transcode* (Section 3). Through these transcoding steps, we generate empirical functions to estimate the b_k, c_k, and s_k values for all replicas. As real-world traffic traces for quality-aware systems are not available, we test various access patterns in our simulations. The simulated video server possesses network bandwidth of 90Mbps (dual T3 lines), four UltraSparc 1.2MHz CPUs, and variable storage capacity (60 to 300G) for data replication. All the above parameters are set to be close to those in a real-world server. We run our experiments on a Sun Workstation with a UltraSparc 1.2MHz CPU.

7.1 Results for Hard Quality Model

In this experiment, we compare our replica selection algorithm (Section 4) 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 2-D space (resolution and frame rate) with 15 to 20 values on each dimension (differs by each video object). Requests (with $f = 10/sec$) are distributed in a Zipf pattern to all M replicas.

In Fig 11, we show the performance of three replica selection methods: our solution that chooses quality points by their f_k/s_k values (Algorithm 1), an algorithm that chooses by their CPU load over storage $\frac{f_k c_k}{\mu_k s_k}$ (Algorithm 2), and one whose selection criteria is access rate (f_k) only (Algorithm 3). The results confirm our analysis in Section 4. Algorithm 1 always gets the lowest reject probability (Fig 11I) as a result of its collecting the highest f_A (Fig 11III). From Fig 11II we can see that the rejection frequency on bandwidth is significantly lower than 1. In these experiments, the recorded total bandwidth loads ($\sum_{k \in A} \frac{f_k b_k}{\mu_k}$) range from 0.65 to 2.75 times of the total bandwidth of 90Mbps. However, the corresponding CPU loads needed for the same traffic are on the order of hundreds of times of the total CPU power. This explains why the observed reject frequency on CPU (Fig 11IV) is always higher than 0.999. As storage increases, P and $P_B^{(c)}$ decrease while $P_A^{(b)}$ and f_A increase. Note when excessive storage is used the decrease of P slows down as bandwidth now becomes the bottleneck.

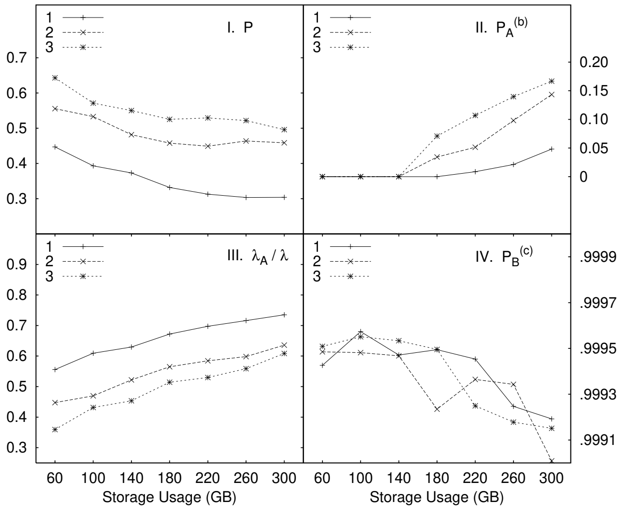


Figure 11: Performance of various replica selection algorithms in the hard quality model.

7.2 Results for Soft quality Model

In this section we present experimental results under the soft quality model. We first evaluate the performance of *Greedy* and *Iterative Greedy* algorithms in terms of optimality (Fig 12) and running time (Fig 13). In this experiment, we set f to 3600 requests/hour so the utility rate is bounded by 3600/hr. We compare our algorithms with three others: 1. the CPEX mathematical programming package[5]; 2. a random algorithm; 3. a *local* algorithm that places replicas in the hottest areas in the quality space. CPLEX is a widely-used software for solving various optimization problems and is well-known for its efficiency. We tune CPLEX such that the results obtained are within a 0.01% gap to the optimal solution.

From Fig 12A, it is clear that our algorithms always find solutions that are very close to the optimal. More details can be found in Fig 12B 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% smaller than the optimal values. The *Iterative Greedy* cuts the gap by at

[5]version 8.0.1, http://www.cplex.com

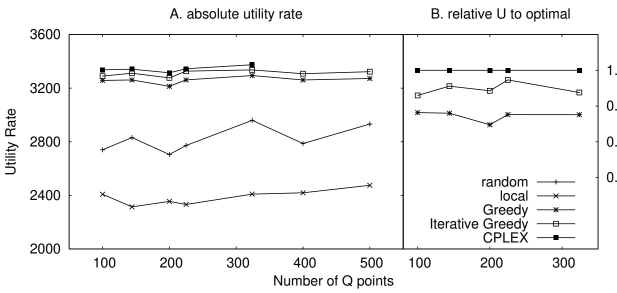


Figure 12: Optimality of replica selection algorithms.

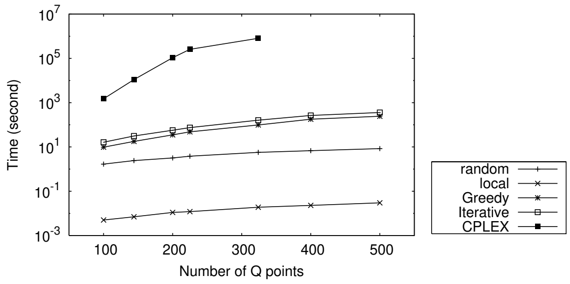


Figure 13: Running time of different replica selection algorithms.

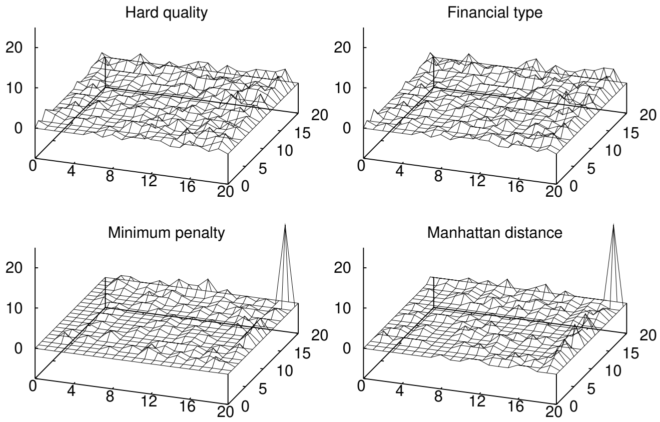


Figure 14: Frequency of replicas chosen in a 20×20 quality space.

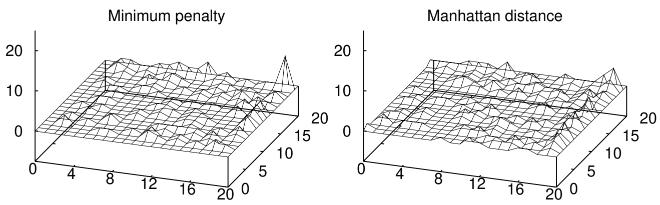


Figure 15: Frequency of replicas chosen by *Iterative Greedy* in a 20×20 quality space.

least half in all cases: its solutions always achieve more than 99% of the optimal utility rate. The performance of both algorithms is not affected by the increase of number of quality points. Nor is it affected by access patterns. We tested different access patterns (e.g. Zipf, 20-80, and uniform) and obtained similar results (data not plotted due to space limitations). 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 4), performs even worse than the random algorithm. This shows that it is dangerous to 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 13. 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. This is good enough for an offline algorithm. We will present the running time of their online versions in Section 7.3.

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. Fig 14 shows the frequency of quality points chosen by *Greedy* in a 20×20 space for a total number of 30 videos. Larger numbers on 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 over 20 videos, *Greedy* picks the lowest quality replica (19, 19). This confirms our discussion in Section 5.3: with overestimated utility rates, smaller replicas are always chosen first. The situation is improved by the *Iterative Greedy* algorithm. Fig 15 shows the distribution of replicas after running *Iterative Greedy* with the same set of inputs. The high peaks on points (19, 19) disappear and total utility rate increases by about 2%.

One thing to point out is 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 5) 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.

7.3 Load Balancing and Dynamic Replication

We study replica selection in a multi-server environment under the hard quality model. Experimental setup is the same as that described in Section 7.1 except the simulator contains 10 identical servers. We compare the performance of three strategies: load balancing by *resource pricing* (Section 6.1), load balancing by *Bandwidth-storage ratio* (BSR) [5], and random assignment of load. Fig 16A shows the results of load balancing using the metric of standard deviation normalized by the mean of loads. The *pricing* strategy has slightly better performance than BSR. The *random* method generates highly unbalanced load distribution. The effect of load balancing on reject rate is presented in Fig 16B: the *random* method performs the worst while *pricing* only has marginal advantages over BSR. From this experiment we conclude that load balancing is needed. However, it is not clear which load balancing strategy is better and further investigation on this is beyond the scope of this paper.

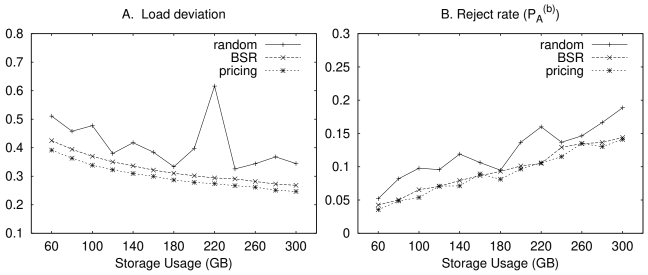


Figure 16: Performance of load balancing methods.

We also 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 match exactly with those found by the modified *Greedy* discussed in Appendix E thus the utility rates are always the same between two solutions. As shown in Fig 17A, the replicas selected by SoftDynaRep have utility rates that are consistently within 99.5% of that by the original *Greedy* algorithm. In this experiment of 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 17B, we record such numbers for each execution of SoftDynaRep and very few of these readings exceeds 15. This shows that our algorithm is suitable for making real-time decisions.

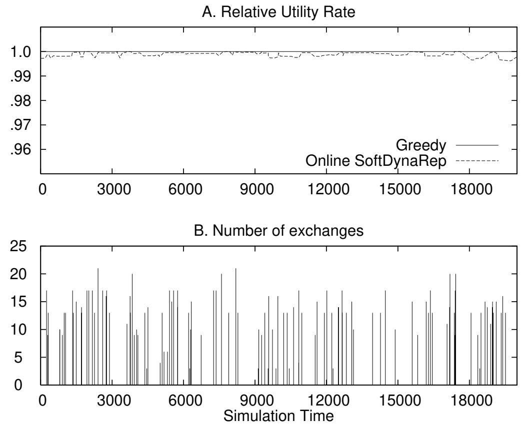


Figure 17: Performance of SoftDynaRep.

8. RELATED WORK

Quality support in media delivery in response to heterogeneous client features and environmental conditions has attracted a lot of attention [19, 18]. However, data replication under storage constraints has not been investigated. Efforts to build quality-aware media systems include [19, 18, 9]. In [24], quality-aware query processing is studied in the context of multimedia databases. They extend the query generation/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 [3] and quality model [25]. None of the above deals with replication of copies with different qualities.

The traditional data replication problem has been studied extensively in the context of web [23, 22], distributed databases [20, 17], and multimedia systems [15, 26]. 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 high acceptance rate and resource utilization by balancing the use of different resources [26, 5, 8]. 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 with data availability. Due to resource constraints, data consistency can sometimes only be enforced loosely. [21] presents a parametric algorithm to control the tradeoffs between data precision and performance under such an *approximate data replication* scheme.

Another important issue is dynamic replication of data. Access frequency 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.* [27] 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 [12]. They found that to take advantages of cached data, it is sometimes necessary to process individual queries using 'sub-optimal' plans in order to reach higher system performance. In [14] and [4], video replication/de-replication is triggered as a result of changes of request rates.

9. CONCLUSIONS

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

10. REFERENCES

[1] E. Amir, S. McCanne, and H. Zhaing. An Application Level Video Gateway. In *Proceedings of ACM Multimedia*, pages 255–265, 1995.

[2] J. Aspens, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-Line Load Balancing with Applications to Machine Scheduling and Virtual Circuit Routing. In *Proceedings of ACM STOC*, pages 623–631, 1993.

[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] C.-F. Chou, L. Golubchik, and J. C. S. Lui. Striping Doesn't Scale: How to Achieve Scalability for Continuous Media Servers with Replication. In *Proceedings of IEEE ICDCS*, pages 64–71, April 2000.

[5] A. Dan and D. Sitaram. An Online Video Placement Policy Based on Bandwidth to Space (BSR). In *Proceedings of ACM SIGMOD*, pages 376–385, 1995.

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

[7] P. Gazdzicki, I. Lambadaris, and R. Mazumdar. Blocking Probabilities for Large Multirate Erlang Loss Systems. *Advances in Applied Probability*, 25:997–1009, December 1993.

[8] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation Algorithms for Data Placement on Parallel Disks. In *Proceedings of SODA*, pages 223–232, 1998.

[9] 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.

[10] O. Kariv and S. L. Hakimi. An Algorithmic Approach to Network Location Problems. II: The p-Medians. *SIAM Journal of Applied Mathematics*, 37(3):539–560.

[11] J. S. Kaufman. Blocking in a Shared Resource Environment. *IEEE Transactions on Communications*, 29(10):1474–1481, October 1981.

[12] D. Kossman, M. Franklin, and G. Drasch. Cache Investment: Integrating Query Optimization and Distributed Data Placement. *ACM Trans. of Database Ssystems*, 25(4):517–558, 2000.

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

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

[15] 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*, 2(6):280–287, January 1995.

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

[17] A. Milo and O. Wolfson. Placement of Replicated Items in Distributed Databases. In *Proceedings of EDBT*, pages 414–427, 1988.

[18] R. Mohan, J. R. Smith, and C.-S. Li. Adapting Multimedia Internet Content for Universal Access. *IEEE Transactions on Multimedia*, 1(1):104–114.

[19] S. Nepal and U. Srinivasan. DAVE: A System for Quality Driven Adaptive Video Delivery. In *Proceedings of Intl. Workshop of Multimedia Information Retrieval (MIR04)*, pages 224–230.

[20] M. Nicola and M. Jarke. Performance Modeling of Distributed and Replicated Databases. *IEEE Trans. Knowledge and Data Engineering*, 12(4):645–672, July/August 2000.

[21] C. Olston, B. T. Loo, and J. Widom. Adaptive Precision Setting for Cached Approximate Values. In *Proceedings of ACM SIGMOD*, pages 355–366, 2001.

[22] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the Placement of Web Server Replicas. In *Proceedings of IEEE INFOCOM*, pages 1587–1596, 2001.

[23] M. Rabinovich. Issues in Web Content Replication. *Data Engineering Bulletin*, 21(4):21–29, 1998.

[24] Y-C. Tu, S. Prabhakar, A. Elmagarmid, and R. Sion. QuaSAQ: An Approach to Enabling End-to-End QoS for Multimedia Databases. In *Proceedings of EDBT*, pages 694–711, March 2004.

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

[26] 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*, 5(5):282–296, September 1997.

[27] O. Wolfson, S. Jajodia, and Y. Huang. An Adaptive Data Replication Algorithm. *ACM Transactions on Database Systems*, 22(4):255–314, June 1997.

APPENDIX

A. THEOREM 4.1

PROOF. The CPU requests from different replicas can be viewed as competitors for a shared resource pool with finite capacity C. The reject probability is studied by a generalization of the famous Erlang loss model [7]. 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 [11], the reject probability of any replica k is

$$P_k = \frac{\sum_{\vec{n} \in S_k} \prod_{k=1}^{M} \frac{1}{n_k!} \left(\frac{f_k}{\mu_k}\right)^{n_k}}{\sum_{\vec{n} \in S} \prod_{k=1}^{M} \frac{1}{n_k!} \left(\frac{f_k}{\mu_k}\right)^{n_k}} \tag{19}$$

where $S_k = \{\vec{n} : C - c_k < \sum_{k=1}^{M} n_k c_k \leq C\}$ and $S = \{\vec{n} : \sum_{k=1}^{M} n_k c_k \leq C\}$ are two sets of states. The states in 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 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 Equation (19). Fortunately, Gazdzicki *et al.* [7] gives the following asymptotic approximation to Equation (19):

$$P_k = \left(1 - e^{\tau c_k}\right)\left(1 + o(1)\right) \tag{20}$$

where τ is the unique solution to the following equation:

$$\sum_{k=1}^{M} \frac{f_k}{\mu_k} c_k e^{\tau c_k} = C. \tag{21}$$

Putting (20) into (21), we get

$$\sum_{k=1}^{M} \frac{f_k c_k}{\mu_k} \left(1 - \frac{P_k}{1 + o(1)}\right) = C.$$

Since $C \ll \sum_{k=1}^{M} \frac{f_k c_k}{\mu_k}$, from the above equation we have

$$\sum_{k=1}^{M} \frac{f_k c_k}{\mu_k} \frac{P_k}{1 + o(1)} = \sum_{k=1}^{M} \frac{f_k c_k}{\mu_k} - C \approx \sum_{k=1}^{M} \frac{f_k c_k}{\mu_k}.$$

The only solution to let the above hold true is $\frac{P_k}{1+o(1)} \approx 1$ for all k. This means $P_k \approx 1$ as P_k can never exceed 1. Immediately, from Equation (5), we get $P \approx 1$. $\square$

B. REJECT PROBABILITY UNDER DIFFERENT LEVELS OF RESOURCE CONGESTION

In addition to Equation (19), the reject probability to a shared resource is also approximated as follows [7]:

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

$$P_k = e^{\tau d \epsilon - I(C)} \frac{d}{\sqrt{2\pi}\sigma} \left(\frac{1 - e^{\tau c_k}}{1 - e^{\tau d}} \right) \big(1 + o(1)\big). \quad (22)$$

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

$$P_k = \sqrt{\frac{2}{\pi}} \frac{c_k}{\sigma} \big(1 + o(1)\big). \quad (23)$$

In equations (22) and (23), τ is defined in Equation (21). Other relevant quantities are defined as follows:

i. d is the greatest common divisor of $c_1, c_2, \ldots, c_M$;

ii. $\epsilon = \frac{C}{d} - \left[\frac{C}{d}\right]$ where $[a]$ denotes the largest integer such that $[a] \leq a$;

iii. $I(C) = \tau C - \sum_{k=1}^{M} \lambda_k \big(e^{\tau c_k} - 1\big)$;

iv. $\sigma^2 = \sum_{k=1}^{M} \lambda_k {c_k}^2 e^{\tau c_k}$.

We first show the reject probability P in light load and critical load situations are asymptotically smaller than 1.

THEOREM B.1. *In our multimedia database system, when there are critical load on a resource, $P = O\big(\sqrt{\frac{2}{\pi}}\big)$.*

PROOF. In this case, $\tau = 0$ and $e^{\tau c_k} = 1$. From Equation (23), we have $\lambda_k P_k^2 = \frac{2}{\pi} \frac{\lambda_k c_k^2}{\sum_{k=1}^{M} \lambda_k c_k^2}$, which leads to

$$\sum_{k=1}^{M} \lambda_k P_k^2 = \frac{2}{\pi}. \quad (24)$$

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

$$L = \sum f_k P_k - \phi \bigg(\sum \frac{f_k}{\mu_k} P_k^2 - \frac{2}{\pi} \bigg)$$

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 Equation (24), get

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

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

$$P = \frac{1}{f} \sum f_k P_k = \frac{1}{f} \sum f_k \mu_k \sqrt{\frac{2}{\pi \theta}} = \sqrt{\frac{2\theta}{\pi f^2}}.$$

It is easy to see that $\theta < f^2$ in our system, we have $P < \sqrt{\frac{2}{\pi}}$. $\square$

It is easy to see that the reject rate in Case 1 is asymptotically smaller than that in Case 2. Now let us turn to the overloading situation (i.e. $\sum \lambda_k c_k > C$). It is difficult, if not impossible, to show the monotonicity of P as a function of the total load. In the following paragraphs, we show that P increases with the increase of total load by making some reasonable assumptions about the query patterns to replicas.

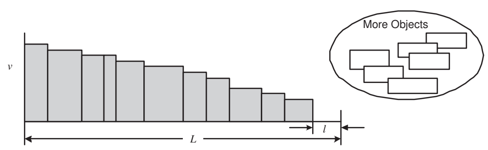
The replica-specific reject probability under the overloading situation is represented in Equation (19). We can see that $\tau < 0$ in this case. For the convenience of discussion, we use $\gamma = -\tau > 0$ instead. In considering the change of P with total load $C' = \sum \lambda_k c_k > C$, we assume that total load change is only caused by the increase of resource requests by each and every traffic class. In other words, the parameters f_k and μ_k are fixed for requests to all replicas. What we are interested in is: when all c_k increase, how does P change? This assumption is totally reasonable in our model with queries accessing two resources: bandwidth (moderately loaded) and CPU (heavily loaded).

When the load increases as all c_k change, it is easy to see that γ increases. From Equation (20), we have

$$P = \frac{1}{f} \sum_{k=1}^{M} f_k \big(1 - e^{-\gamma c_k}\big).$$

For each k, $f_k \big(1 - e^{-\gamma c_k}\big)$ increase as both γ and c_k increase. As a result, P also increases.

C. OPTIMALITY OF A HEURISTIC ALGORITHM TO SOLVE 0-1 KNAPSACK



As illustrated in the graph above, the Knapsack has size L, Y axle represents the value density (denoted as v). The Knapsack is filled with objects with the largest v. It is easy to see that, if all L storage is filled, the solution is optimal as any other plan will decrease total value. If there is a unfilled region with size l, we can fill it with the largest density v' among the unselected objects. This generates an infeasible solution but it gives an upper bound of the optimal total value: $\hat{v} = \Omega + lv'$ where Ω is the area of the shaded region. Here lv' can be viewed as the gap between our solution and the optimal value. When $L \gg l$, we have $lv' \ll \hat{v}$.

D. LOAD BALANCING IN DISTRIBUTED DATA REPLICATION

Suppose set A contains certain number of replicas and they are to be placed on N servers. Denote the total query rate in server i as f_i and reject probability as $P_i^{(b)}$. Immediately, we have $f_A = \sum_{i=1}^{N} f_i$ and $P_A^{(b)} = \frac{1}{f_A}\sum_{i=1}^{N} P_i^{(b)}$. In distributed replication, we have the problem of *minimizing* $P_A^{(b)}$ *given* f_A, which can be solved by using Lagrangian multipliers with the following solution:

$$\frac{\partial}{\partial f_i}\bigg(\sum_{i=1}^{N} f_i P_i^{(b)} - \phi\big(\sum_{i=1}^{N} f_i - f_A\big)\bigg) = 0, \quad \forall f_i \qquad (25)$$

where ϕ is a Lagrange multiplier. For any i, the LHS of Equation (25) is $P_i^{(b)} - \phi$. Thus, we get the following condition of minimality

$$P_1^{(b)} = P_2^{(b)} = \cdots = P_N^{(b)} = \phi. \qquad (26)$$

Theoretically, it is not clear how to achieve uniform $P_i^{(b)}$ in our case. Little *et al.* [15] proved that, when requests have the same bandwidth requirements, the above condition is achieved when all servers have the same load. Here we conjecture that this is still true in our system where requests have heterogeneous bandwidth requirements. We verify it by experiments.

E. OPTIMALITY OF SOFTDYNAREP

To show the quality of replica selection of SoftDynaRep is as good as that of rerunning Greedy, it is necessary to investigate more details of the algorithm. Line 9 to line 12 of Fig 10 can be represented in Fig 18. 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 $k > 0$).

Now we compare the replica selection of SoftDynaRep and *Greedy*. From discussions in Section 6.2.2, we understand that the global RR changes as the query rates of individual replicas change and Greedy (implicitly) rebuilds the global RR. Essentially, *Greedy* selects those replicas with the largest utility density on the global RR, similar to the solution for 0-1 Knapsack shown in Appendix C. We first consider a modified version of *Greedy* with a subtle difference from the one represented in Fig 5: we replicate items along the global RR till we encounter the first replica k' that cannot be accommodated by the available storage (equivalent to l in Appendix C). The original *Greedy* algorithm is smarter than this: it will try to fill l with replicas with lower utility density than k'. Thus, the replicas selected by the modified *Greedy* is a consecutive chunk of the global RR (from the beginning to the one prior to k') while those selected by the original version may have holes in it. Due to the reason discussed in Appendix C, the total utility rate achieved by this modified version is only slightly smaller than that of the original version. We have the following lemma showing that the replica selection by SoftDynaRep is at least as good as the modified *Greedy*:

```
1  storage ← available storage
2  k ← 0, j ← V − 1
3  while k ≤ 0
4  do     r0 ← flist[k]
5         victims ← ∅
6         while storage < size of replica r0
7         do     r1 ← blist[j]
8                if r0 and r1 belong to the same video
9                   j ← j − 1
10                  continue
11               if utility density of r1 > utility density of r0
12                  k ← k + 1
13                  rollback blist to its status on line 6
14                  break
15               else append r1 to victims
16                    update and sort blist
17        if storage ≥ size of replica r0
18           Exchange (r0, victims)
19           update and sort both flist and blist
```

Figure 18: Selection of replicas for storage exchange in SoftDynaRep.

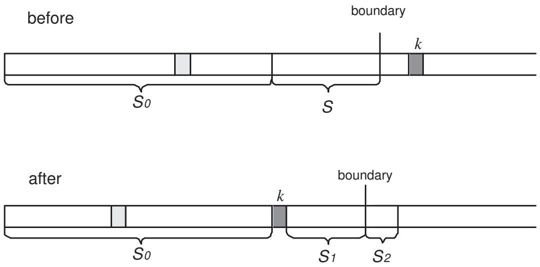


Figure 19: Replica selection upon query rate change.

Lemma 2. *With the same replica-specific inputs and change of query rate of a specific video, if a replica is selected by the modified version of* 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 Figure 19, 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 till 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 query rate increase, some replicas of v (represented as shaded boxes) 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 modified *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 *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; 2. storage is taken from replicas with utility density 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 directly go to line 17 in Fig 18 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 are 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 19). These replicas are not affected by SoftDynaRep as we never sacrifice such replicas for k (line 11, Fig 18).

Case 2.2. Replicas whose utility density is smaller than that of k (e.g. those in region S_1 in Fig 19). These replicas are part of region S before the query rate change. One nice feature of the modified *Greedy* is that all replicas chosen is 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 the list *victims* is also a consecutive chunk as it is formed by always choosing the replica with the smallest utility density, starting (backwards) 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. $\square$