2004

Quality-Aware Replication of Multimedia Data

Yi-Cheng Tu

Sunil Prabhakar
Purdue University, sunil@cs.purdue.edu

Report Number:

04-023

QUALITY-AWARE REPLICATION OF MULTIMEDIA DATA

Yi-Cheng Tu
Sunil Prabhakar

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #04-023
July 2004

Quality-Aware Replication of Multimedia Data

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

Abstract

Quality-aware multimedia systems adapt media transmission to user QoS requirements. Such adaptation is generally performed by transcoding original (high-quality) media into quality-heterogeneous copies (replicas). As media transcoding is extremely expensive in terms of CPU cost, the strategy of static adaptation by which replicas are pre-encoded and stored in disk has been widely exploited. However, the problem of selecting which replicas to store from the large number of candidates has not been studied. We view this as a data replication problem in a virtual quality space. In this paper, we give (heuristic) solutions to this NP–complete problem under two different system models. For a system that only entertains hard quality requirements, we formulate the problem into a 0–1 Knapsack and propose an efficient solution that minimizes request blocking probability. More interesting is a soft quality-aware system where users are willing to negotiate their QoS needs. An important optimization goal is to minimize utility loss in such systems. We propose a powerful algorithm, the Iterative Greedy algorithm, to solve this optimization, which is found to be a variation of the k-median problem. Extensive simulations show that our algorithm performs significantly better than other widely-used heuristics. Most of the results given by the Iterative Greedy algorithm are very close to the optimal value. In many cases, it finds the optimal solution.

1. Introduction

Due to the streaming fashion of data delivery, multimedia applications place high demands on Quality-of-Service (QoS) and reliability. Support of user QoS requirements is critical in building media services as (human) user satisfaction is the primary factor that determines the success/survival of such services. Quality-aware multimedia systems [28, 22, 10] allow users specify the *quality* [1] of the media to be delivered. Quality requirements are very user-specific: they depend on user's practical needs and resource availability on the client devices [22, 20]. The quality parameters of interest also differ by the type of media we are dealing with. For digital videos, the quality parameters of interest include resolution, frame rate, color depth, signal-to-noise ratio (SNR), audio quality, compression format, and security level [28].

From the point of view of video servers, satisfying user quality specifications requires (operating) system and network support and careful design of high-level adaptation mechanisms. Such adaptation is generally accomplished in two complementing ways: *dynamic adaptation* and *static adaptation* [22]. In dynamic adaptation, media data are transcoded (based on a high-quality copy) at runtime to the appropriate quality and format required by the users. The problem for dynamic adaptation is: transcoding is very expensive in terms of CPU cost therefore online transcoding is difficult in a multi-user environment. Our experiments (Fig 1) running 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 (depending on the resolution). This corresponds to 0.6 to 2.4 times of the entire CPU power if the frame rate for the video is 25fps. As a result, many *transcode proxy* [2, 7] servers or video gateways [1] with massive computing power have to be deployed. Static adaptation attempts to solve this problem by storing precoded quality-heterogeneous copies of the original media on magnetic disks. By this, the heavy demand on CPU power at runtime is alleviated. Apparently, we are trading disk space for CPU cycles, which is a cost-effective way to scale as disks are much cheaper than CPUs.

Current researches on adaptive media systems focus on topics such as quality specification, transcoding algorithms, and system architecture. Data placement under storage constraints is never investigated. Even for systems that perform

1 Here *quality* refers to user-level QoS. In this paper, the words *QoS* and *quality* are used interchangeably.

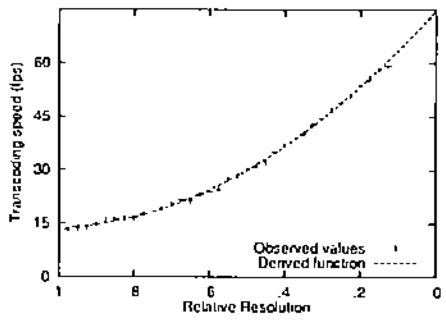

Figure 1. Speed of transcoding a 640×480 MPEG1 video to various resolutions.

static adaptation, they are designed under the assumptions of either *there is always enough storage space* or *user requests concentrate on a small number of quality profiles*. However, from our experience of building a quality-aware video DBMS [28], storage space is a concern for performing static adaptation. First of all, users vary widely in their quality needs and resource availability [20]. This leads to a large number of quality-specific copies of the same media content to be stored in disk. Secondly, although cheap, storage space is not free. For a commercial media streaming service, adding storage is not as simple as buying disks for our home PC due to the high requirements on reliability. Most likely they lease all their resources from vendors such as Akamai to form a Content Distribution Network (CDN). With these concerns on (monetary) costs, the extra disk space used for static adaptation should not grow unboundedly. An analysis in Section 4 shows the disk space needed to accommodate all possible quality profiles could be intolerably high. Therefore, the choice of quality-heterogeneous media copies to store becomes important and is the very problem we try to solve in this research.

We view the selection of media copies for storage as a data replication problem. Traditional data replication focuses on placement of copies of data in various nodes in a distributed environment [17]. 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 quality-heterogeneous copies generated via transcoding. In spite of the differences in problem definition, techniques can be borrowed to tackle our problem. In this paper, we present strategies to smartly choose quality of replicas under two different assumptions about user behaviors. Important performance metrics include:*acceptance rate* of requests, *user satisfaction* , and *resource consumption* [4, 13]. Our data replication algorithms are designed to achieve the highest acceptance rate or user satisfaction under fixed resource (CPU, bandwidth, and storage) capacities. The major contributions of this paper are:

1. In a hard quality-aware model where users are assumed to be strict on quality, we develop a replica selection algorithm that optimizes system acceptance rate;
2. We formulate the replica placement under a soft quality-aware model as a k-median problem with the goal of maximizing user satisfaction. We evaluate various heuristics and comment on their applicability and performance in our problem;
3. We propose and test a new algorithm, the *Iterative Greedy* algorithm, to solve the problem mentioned in 2. Without increasing time complexity, our algorithm always finds better solutions than other heuristics. In many cases, it finds the optimal solution.

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 media services. The fact that our *Iterative Greedy* algorithm frequently finds the solution of a NP-complete problem may also lead to further exploration on the theoretical side.

The remainder of this paper is organized as follows: we first describe related work in Section 2; in Section 3 we introduce the system model and the assumptions we make; Section 4 discusses storage use of the replication process; we present our replica placement algorithms in Section 5 and Section 6; Section 7 is dedicated to experimental results; we conclude the paper by Section 8.

2. Related Work

Adaptation of media delivery in response to heterogeneous client features and changing environmental conditions has attracted a lot of attention from both multimedia and network communities [22, 14, 20, 6]. The efforts to build quality-aware media systems include [22, 20, 10]. Our previous work [28] utilizes the same idea in the context of multimedia databases. Two other QoS works in multimedia databases discuss quality specification [3] and QoS model [29]. None of the above has concerns on data placement of copies with different qualities. The closest work in quality-aware data replication is by Steinmetz and coworkers [24]. They focus more on availability and consistency of non-media data.

The traditional data replication problem has been studied extensively in the context of web [17, 27], multimedia systems [16, 30], and distributed databases [23, 19]. 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 is a major concern in early multimedia systems. A number of algorithms are proposed to achieve high acceptance rate and resource utilization by balancing the use of different resources [30, 5, 9]. 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.

Another important issue is dynamic replication of data. Access frequency to individual data items are likely to be changing in most environments. How to make the replication strategy quickly and accurately adapt to these change of access patterns is not a trivial problem. Wolfson *et al.* [31] introduced an algorithm that changes the location of replicas in response to changes of read-write patterns of data items. In [15] and [4], video replication/de-replication is triggered as a result of changes of request rates.

In [26], the placement of multi-resolution images on parallel disks was studied with the goal of minimizing I/O costs.

3. System Model and Assumptions

This research is based on a generic CDN-like architecture such as the one described in [4]. The system consists of a number of *servers* on the edge of the Internet. All servers are connected to a *Global Switch* via high-capacity links. User requests are sent to the nearest local server for service.

Before going into more details of the model, we list some of the notations that will be used throughout this paper:

Symbol	Definition
B	Total server bandwidth
S	Total server disk space for storing media
C	Total server CPU power
V	Number of media objects in the system
M_i	Number of points in the quality space of a media i
M	Total number of quality points for all media
λ_k	arrival rate, average number of requests per unit time
μ_k	service rate, average number of requests served per unit time
c_k	CPU cycles for online transcoding into this Q point
b_k	bandwidth needed for streaming
s_k	storage space needed if a replica is placed

A server is an entity with finite computing resources. In our model, the servers are characterized by the total amount of the following resources they carry: 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. With striping disk farms deployed, network bandwidth is most likely to be the bottleneck in modern media servers [25].

The system contains V media objects. User requests come with an ID of the object to be retrieved as well as 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 (called as *quality point* or *Q point* thereafter) in a m-dimensional space. Generally, a quality parameter can only take some discrete values within a range along a quality dimension. For example, the spatial resolution of a video is expressed as a (natural) number of pixels within the range of 192×144 (low-quality MPEG1) to 1920×1080 (HDTV). Furthermore, the (horizontal) resolution can only be multiplies of 16 as the latter is the finest granularity most transcoders can handle. The total number of Q 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 Q point is a candidate for replica placement.

We model the requests to media objects with a quality vector as traffic classes. Any traffic class k among the $M = \sum_{i=1}^{V} M_i$ classes is characterized by parameters λ_k, μ_k, c_k, b_k, and s_k. For any class k, we assume the arrival of requests is a Poisson process with parameter λ_k and the duration of streaming sessions follows 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. A streaming session could be arbitrarily long in time as the users may use VCR functionalities (e.g. stop, fast forward/backward) during media playback. The total request rate for all classes is $\lambda = \sum_{k=1}^{M} \lambda_k$. The last three ($c_k, s_k, b_k$) parameters correspond to usage of resources. They can be (precisely) estimated from empirical functions derived by regression (see Section 4). Note c_k is fixed for a class k 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) transcodes a copy from a high-quality replica (by consuming c_k units of CPU) if the corresponding replica does not exist;
3) rejects the request if not enough CPU is available.

If either 1) or 2) is performed, retrieved/transcoded media data is transmitted to the client via the network (using b_k units of bandwidth). We assume the CPU cost of operations other than transcoding is zero. This is reasonable in the context of this research as those costs are trivial comparing to transcoding and do not change with the specified q. In the above model, requests are either admitted or rejected. In Section 6, we will study a more flexible model where users may compromise the original quality they specify.

3.1. Assumptions

As the first research addressing the problem, we focus on issues in a single server. Thus, we assume requests are only served in the local server. In practice, the requests are directed to other servers when the local server does not have enough resource. Extensions that take all nodes into account will be presented in a follow-up paper.

The paper deals with the situation of *static* data replication, which means the inputs (traffic class parameters and server resource) to our algorithm do not change. The importance of studying static replication is justified by the follows: 1. access patterns to media systems, especially video-on-demand systems, remain the same within a 24-hour period [16]; 2. conclusions drawn from static replication form the basis for dynamic replication research [15].

4. Storage Occupation for Replication

As mentioned in Section 1, in previous works it is assumed enough storage is available for static adaptation. As user quality requirement could hit any quality point in the space, an ideal solution is to store most, if not all, of these replicas in disk so only minimal load is put to the CPU for online transcoding. In this section, we will show that the storage cost for such a solution is simply too high.

We use digital video as an example in the following discussion. From empirical equations we derived to estimate the storage consumption of QoS-specific replication, we can estimate the storage occupation for QoS-specific replication. According to [22], the bitrate of a video replica with a single reduced QoS 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

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

Table 1. Total relative storage vs. N.

the bitrate of all copies can be obtained by:

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

Storage occupation can be easily calculated as $TF_0\frac{N}{\beta+1}$ where T is the playback time of the media. Note the above only considers one quality dimension. In [22], Equation (1) is also extended to three dimensions (spatial resolution, temporal resolution, and SNR):

$$F = \alpha F_0(1 - R_A^{\frac{1}{\beta}})(1 - R_B^{\frac{1}{\gamma}})(1 - R_C^{\frac{1}{\theta}}) \qquad (2)$$

where R_A, R_B, and R_C are quality change in the three dimensions, respectively. They also post the constants obtained from their transcoder(s): $\alpha = 1.12$, $\beta = 1.5$, $\gamma = 1.7$, and $\theta = 1.0$. Using the same technique of approximation by integration as used above, we can easily see the sum of all storage needed for all N^3 replicas is $TF_0O(N^3)$ (details skipped). What this means is: the relative storage (to original size) needed for static adaptation is in the order of total quality points. The latter depends on the density of replication (N) along the quality dimensions. Some of the storage costs generated using Equation 2 are listed in Table 1. For example, when $N = 10$, the extra storage needed for all replicas is $117.7 - 1 = 107.7$ times of the original media size. These numbers are roughly on the neighborhood of $0.1N^3$. 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 also test the equations generated in [22] by our own experiments. We use the open-source video processing tool named *transcode* [2] in these experiments. Fig 2 shows the relative video size when spatial resolution is degraded into various percentages. The points are the video size we observed and the curve $1 - R^{1/1.5}$ represents Equation 1. We also plot a straight line $1 - 1.25R$. From regression, the observed values can be closely matched by a polynomial func-

2 http://www.theorie.physik.uni-goettingen.de/ ostreich/transcode/

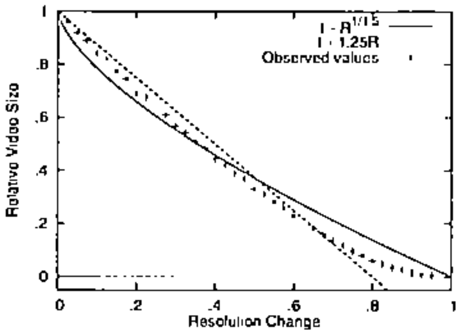

Figure 2. Change of video bandwidth with resolution degradation.

tion $0.421R^3 - 0.003R^2 - 1.419R + 0.989$. In this graph, the areas under the curves can be viewed as the total relative storage use. We know the area of the triangle by the X, Y axles and the line $1 - 1.25R$ is $\frac{2}{5}$, exactly the number given by $\frac{N}{\beta+1}$ as $\beta = \frac{3}{2}$. We can see that the areas under three curves are roughly the same.

5. Hard Quality-Aware Systems

In this section, we discuss data replication strategies in hard quality-aware systems where users have rigid QoS requirements on media streaming. This means the users are not willing to negotiate when the QoS he/she specifies cannot be satisfied due to resource congestion. As mentioned in Section 1, the main idea of static adaptation is to replicate original media into quality-heterogeneous copies such that the demand on CPU cycles (most stringent resource) decreases. We also show that we do not have access to enough disk capacity to cover 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. As a request is either rejected or accepted, we use *blocking (reject) probability* as the metric for performance evaluation.

In our system, the reject rate of requests are determined by the runtime availability of two resources: bandwidth and CPU. Thus, the blocking probability for all requests is

$$P = 1 - (1 - P_c)(1 - P_b)$$

where P_c and P_b are the probabilities of blocking due to the lack of CPU and bandwidth, respectively. We can see that, in order to minimize P, our replica placement policy should set a target to minimizing P_c. P_b is not affected because the bandwidth use for a replica is always the same regardless of what replicas are stored. In an extreme case with unlimited storage thus all M copies are replicated, P_c becomes zero. Let the output of the replica placement algorithm be a vector $R = (r_1, r_2, ..., r_M)$ with 0/1 elements ($r_k = 1$ if replica k is to be stored in disk). Formally, the replica placement problem is to

minimize P_c
given $\quad \sum_{i=1}^{M} r_k s_k \leq S$.

The λ_k, μ_k, s_k, and c_k values for all classes are known. To further approach the problem, we need to find the relationship between P_c and the above class-specific values. First of all, it is easy to see that the overall blocking probability of all classes is

$$P_c = \sum_{k=1}^{M} \frac{\lambda_k}{\sum_{k=1}^{M} \lambda_k} P_k = \sum_{k=1}^{M} \frac{\lambda_k}{\lambda} P_k. \tag{3}$$

where P_k is the blocking probability of class k.

We also understand that P_k is a function of λ_k, μ_k, and c_k for all M classes. The different classes of requests can be viewed as competitors for a shared resource pool (CPU in our case) with finite capacity. The blocking probability is studied using a generalization of the famous Erlang loss model [8]. 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 class k requests currently being serviced. According to [12], the blocking probability of any traffic class k is

$$P_k = \frac{\sum_{\vec{n} \in S_k} \prod_{k=1}^{M} \frac{1}{n_k!} \left(\frac{\lambda_k}{\mu_k}\right)^{n_k}}{\sum_{\vec{n} \in S} \prod_{k=1}^{M} \frac{1}{n_k!} \left(\frac{\lambda_k}{\mu_k}\right)^{n_k}} \tag{4}$$

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 class k will be blocked (as there are less than c_k units of resource available) while S is the collection of all possible states.

Qualitative analysis (Appendix A) of Equation (4) shows that P_k increases with the increase of λ_k, c_k and the decrease of μ_k. This implies that, to decrease blocking probability, we should put as little load ($\frac{\lambda_k}{\mu_k} c_k$) as possible to the CPU. Based on this, an immediate solution would be to sort the traffic classes by their loads and pick the ones with the largest loads for replication. To deal with the storage constraints, we can sort them by $\frac{\lambda_k c_k}{\mu_k s_k}$ (i.e. load divided by storage cost). Is our intuition correct?

Suppose, by determining which replicas to store, our algorithm divides the M classes into two disjoint sets: A containing replicated classes and B containing non-replicated classes. From formula (3), P_c can be expressed as

$$P_c = \frac{\lambda_A P_A + \lambda_B P_B}{\lambda} = \frac{\lambda_B P_B}{\lambda} \tag{5}$$

where $\lambda_A = \sum_{i \in A} \lambda_i$ is the total request rate in set A, P_A the blocking probability of all requests from A and λ_B, P_B

are their counterparts in set B. As no blocking on CPU will occur when a replica is placed, we have $P_A = 0$. Now things become more clear: to get the optimal P_c, we need to minimize $\lambda_B P_B$. The intuitive approach mentioned earlier that chooses replicas by CPU load may work better in optimizing P_B but our interest is in P_c. By putting a class k into A, we get a net decrease of λ_k/λ from P_c. On the contrary, if we use the intuitive algorithm, we may slightly decrease P_B, but that change has to overcome the increase of λ_B to impact P_c (since we get the smallest λ_B only if we select Q points by their λ_k values). In practice, as transcoding jobs are so expensive and storage space for replication is limited, the CPU load in set B is high no matter how we select B. The following theorem shows that P_B is always close to 1 under such circumstances. Therefore, to get a smaller P_c value, we only need to make λ_A (λ_B) as big (small) as possible.

Theorem 5.1 *For a group of M Poisson-arrival traffic classes, if $\sum_{i=1}^{M} \frac{\lambda_i c_i}{\mu_i} \gg C$, $P \approx 1$. All notations follow the same definitions in Section 3.*

Proof: See Appendix B. □

The rest part of the problem becomes easy: to get as large (small) a λ_A (λ_B) value as possible given the storage constraints. This is obviously a 0-1 Knapsack problem. A good heuristic is as follows: sort all possible replicas by their request rate per unit storage occupation (λ_k/s_k) and select those with the highest such values till the total storage is filled – a $O(M \log M)$ algorithm. As we can safely assume $S \gg s_k$ for all $k \in [1, M]$, the heuristic should produce fairly good results.

6. Soft Quality-Aware Systems

In our discussions on the 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 QoS are either strictly satisfied or the request is not serviced at all. However, human users can tolerate some changes of quality [22]. In designing media delivery systems, we can further assume users are flexible in the quality they are actually looking for. The quality parameters they send in along with the requests represent the most desirable situation. 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 (usually degraded) set of quality parameters is called *renegotiation* [28]. 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.

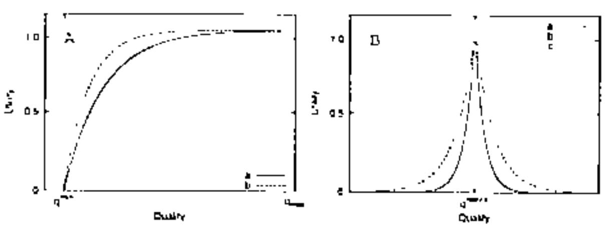

Figure 3. Different types of utility functions.

6.1. Utility Functions

We generally use *utility* to quantify human happiness or satisfaction on a service received [18]. For our purposes, *utility functions* can be used to map quality to utility (as a real number in [0, 1]) and the penalty applied to the media service due to renegotiation is captured by *utility loss*. As utility directly reflects the level of satisfaction from users, it is an important optimization goal in quality-critical systems [13]. We thus set the main goal of our replica placement strategies to be minimizing utility loss. In the soft quality model, increasing utility and decreasing blocking rate are two conflicting goals. We sure can study the trade-off between these two by putting different weights on them but this paper. The server operations shown in Section 3 also need to be changed in soft quality systems: in case of a miss in step 1), the server chooses a replica that yields the smallest utility loss to retrieve. The request is rejected only when bandwidth is not sufficient. For simplicity, we skip the 'negotiation' process between client/server and let the server make the decision.

Figure 3 shows various types of utility functions for a single quality dimension. In general, utility functions are convex monotones (Fig 3A) due to the fact that users are always happy to get a high-quality service, even if the quality exceeds his/her needs. This makes our replica selection a trivial problem: always keep the one with the highest quality. However, in a more realistic environment, the extra quality comes with a cost: more resource have to be consumed (Section 1) and this negatively affects utility under excessively high quality. Taking this into account, we propose a new group of utility functions in 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 3B). The pattern of utility decrease with change of quality can either be dramatic (a, b of Fig 3B) or uniform (c of Fig 3B). Note the functions do not have to be symmetric on both sides of q^{desire}. The hard quality-aware model in Section 5 can be viewed as a special case of the soft quality system: its utility function takes the value of 1 at q^{desire} and 0 otherwise.

The functions mentioned above are for one single quality dimension only. The utility for a quality vector with multiple dimensions is generally given as a weighted sum of dimensional utility described above [13]. The weights of individual quality dimensions are also user-dependent.

6.2. Data Replication as a k-Median Problem.

Let us first study the replica placement problem in the context of one single media object i with M_i quality points. At this point, we assume we know that K ($K > 1$) replicas should be placed in the quality space for i. The determination of K for all V media objects is another interesting optimization problem. However, this is beyond the scope of this paper and we leave it as future work.

Formally, the replica placement problem is to pick a set L containing K points that gives the smallest total *utility loss rate*

$$U = \sum_{k=1}^{M_i} \lambda_k u(k, L)$$

where $u(k, L)$ gives the smallest utility loss when a request to point k is served by retrieving a replica in L. Obviously, we have $u(k, L) = 0$ if $k \subset L$. We know $u(k, L)$ is a function of the distance between k and its nearest neighbor in L (Section 6.1). For example, when we put equal weights on both quality dimensions in a 2-D space and use linear functions such as c in Fig 3B, $u(k, L)$ is actually the Manhattan distance between two points. We weight the utility by the request rate λ_k. The optimal placement of replicas turns out to be a version of the k-median problem [11, 27].

The original k-median problem is stated as follows: given a network of N nodes, we are to select K ($K < N$) nodes as service centers. Each node k requests service at rate λ_k and there is a cost $d_{ki}\lambda_k$ associated with service provided by node i where d_{ki} is the distance between nodes i and k. Each node get service from the closest center. The goal is to minimize total cost.

The k-median problem is proved to be NP–complete [11]. Qiu *et al.* [27] propose a number of heuristics to look for optimal locations for web caching and replication. We will revisit two of these algorithms: the *Greedy* and *Hot Area* algorithms in this section and evaluate their performance and applicability to our replica placement problem in Section 7. We also present a new heuristic, the *Iterative Greedy* algorithm, which has significantly better performance than *Greedy* and *Hot Area*.

6.2.1. The *Greedy* Algorithm.

The main idea of *Greedy* algorithm is to aggressively choose replicas one by one. The first replica is assigned to a point that yields the smallest U as if only one replica is to be placed. This can be done in $O(M_i^2)$ time. The following replicas are determined in the same way with knowledge of previously selected replicas. The time complexity for *Greedy* is $O(K^2 M_i^2)$ for a media object i. It is not $O(KM_i^2)$ as in [27] because in choosing the k-th replica, the distance of any point to a candidate is given by looking at all k chosen points for the shortest distance. So we have complexity $O(M_i^2 + 2M_i^2 + \cdots + KM_i^2) = O(K^2 M_i^2)$. More details of *Greedy* can be found in Appendix C.

6.2.2. The *Hot Area* Algorithm.

This algorithm gives high priority to areas with highest request rates in the metric space. As technical details of *Hot Area* algorithm is not shown in [27], we present one implementation of the algorithm in Appendix C. The algorithm runs the procedure HOTAREA for a few iterations with different *size* inputs and we pick the one that gives the best solution. The *size* parameter represents the size of the potential hot areas we are looking for. For example, when *size* is set to 3, we sort all cubes with side length *size* by their total request rates and put replicas in the first K cubes. The choice of *size* just come out of experience. Comparing to *Greedy*, *Hot Area* is more 'greedy' in the sense that it only considers local optimization by putting replicas directly in the hottest regions. The relative location of hot regions and other cold regions are never considered. In utilizing this algorithm, we expect (assume) the existence of some hot areas in the metric space. The time complexity of procedure HOTAREA is $O(M_i \log M_i + size^2 M_i)$ and we usually run it for a constant number of iterations.

6.3. The *Iterative Greedy* Algorithm

The idea of *Iterative Greedy* algorithm is based on an attempt to improve the performance of *Greedy*. We notice that at each step of *Greedy*, some local optimization is achieved: the $(k+1)$-th $(k+1 \leq K)$ replica chosen *is* the best given the first k replicas. The problem is: we do not know the correctness of the previously selected k replicas. However, we believe the $(k+1)$–th replica added is more 'reliable' than the previously selected k replicas because more global information (existence of other selected replicas) is leveraged in the selection process. Although these information might also be incorrect, the awareness of such existence gives us more confidence on the 'quality' of the selected replica. An extreme case is the first one: it is chosen taking no global information (future replicas) into account. Based on this conjecture, we develop the *Iterative Greedy* algorithm that iteratively improves the 'correctness' of the replicas chosen. The basic idea is to repeatedly get rid of the most 'unreliable' replica and replace it with a new one with the other $k-1$ replicas unchanged. One thing we are sure about is: when we take one replica out, the next one we get will never be worse due to the local optimization we mentioned above. This mechanism resembles that of a genetic algorithm ex-

```
REINSERTION (slist, K, I)
1 Let rlist be an integer list with length $K$
2 $u_{min} \leftarrow$ total utility loss rate of $slist$
3 for $i \leftarrow 0$ to $I$
4     do $slist[i \bmod K] \leftarrow slist[K-1]$
5         $u \leftarrow$ ADD-REPLICA($slist, K-1$)
6         if $u < u_{min}$
7             $u_{min} \leftarrow u$
8             $rlist \leftarrow slist$
9 return $rlist$
```

cept the new (hopefully improved) solution is not chosen from random mutations.

The operations in *Iterative Greedy* are straightforward: all selected replicas are stored in a FIFO queue *slist*. For every iteration we dequeue *slist* and find one replica based on what are left. The newly identified replica is then added to the tail of *slist* if it improves the performance. The main procedure of *Iterative Greedy*, REINSERTION, is shown in the following box. The inputs to REINSERTION include a list of initially selected replicas *slist*, number of replicas to place K, and the number of iterations I. We use a circular array for the queue. The subroutine ADD-REPLICA finds the best replica given the first $K-1$ replicas in the list, put the newly-found in $slist[K-1]$ and return the U value of the selected group of K replicas. Note ADD-REPLICA is also the main body of *Greedy* (Appendix C).

Iterative Greedy is not an extension to *Greedy* only: it can be applied to any initial list of replicas. As to the number of iterations I, we could either repeat REINSERTION until the result converges or fix the number I. Our experiments (Section 7.2.4) show that the result converges after $O(K)$ iterations in most cases. ADD-REPLICA runs at $O(KM_i^2)$. Therefore, total time complexity of *Iterative Greedy* is $O(IKM_i^2) = O(K^2M_i^2)$, same as that of *Greedy*.

7. Performance Evaluation

We study the behaviors of various algorithms described in previous sections by simulations. Our simulated multimedia streaming server contains real file traces of 540 digital videos extracted from the VDBMS video database [3]. We set the μ_k values for all classes to be slightly smaller than the standard playback time of the corresponding media. Some of the raw videos are then transcoded into replicas of different spatial resolution and frame rates using *transcode* (Section 4). By these transcoding practices, we generate empirical functions to estimate the b_k, c_k, and s_k values for all replicas. 2-D spaces are utilized for the simplicity of data presentation. We do not see any reason why our algorithm shall not work for more dimensions. Our experience shows that these estimations are very precise using a reasonable number of samples. We use various synthetic traffic patterns in our simulation as real-world access traces to quality-heterogeneous copies of videos are not available. The simulated video server carries network bandwidth of 90Mbps (roughly the bandwidth of dual T3 lines), four UltraSparc 1.2MHz CPUs, and variable storage capacity (60 to 140G) for replications.

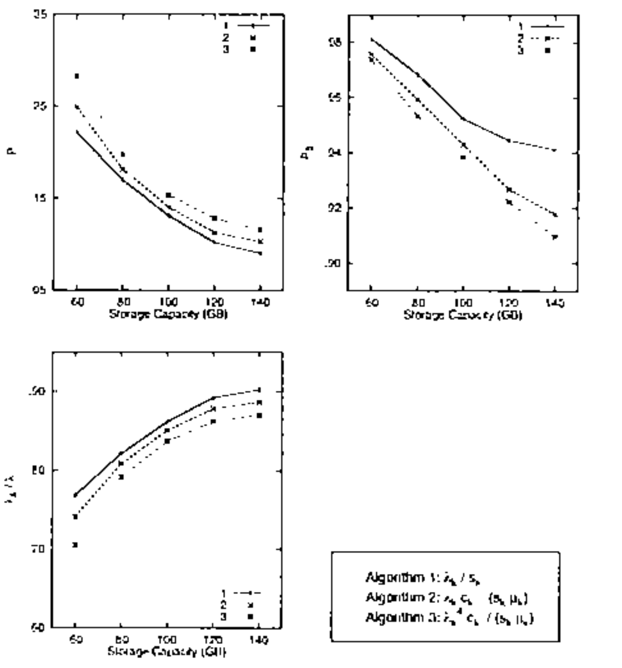

Figure 4. Performance of various selection rules in the hard quality-aware model.

7.1. Reject Rate in Hard Quality Model

In this experiment, we compare our replica selection algorithm (Section 5) to various heuristics under the hard quality model. The metric is the reject (blocking) frequency measured as the ratio of the total number of rejected requests to total requests. The quality space is a 2-D space (representing resolution and frame rate) with 15 to 20 values on each dimension (differs by each video object). Total request rate (λ) is set to 2000 requests/hour and λ is distributed in a Zipf pattern to all M replicas.

In Fig 4, we show the blocking rates of three replica selection algorithms: our solution that chooses quality points by their λ_k/s_k values (Algorithm 1), the intuitive algorithm

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

that chooses by their CPU load $\frac{\lambda_k c_k}{\mu_k s_k}$ (Algorithm 2). and a variation of the intuitive algorithm whose selection criteria is $\frac{\lambda_k^\alpha c_k}{\mu_k s_k}$ (Algorithm 3). The reason for testing the third algorithm is that the when we select replicas by $\frac{\lambda_k^\alpha c_k}{\mu_k s_k}$ $(\alpha > 1)$. we get better P_B than the intuitive algorithm $(\alpha = 1)$ (see Appendix A). We compared the P. P_B. and λ_A values of these algorithms under different storage constraints.

The results confirm our analysis in Section 5. Algorithm 1 always gets the lowest blocking probability P, followed by Algorithm 2 and 3. The P_B values achieved by various algorithms are in an opposite order: algorithm 1 is the worst performer. This can be explained by the λ_A data. Algorithm 1 gets the highest λ_A. or the lowest λ_B in other words. This means that λ_B is the dominant factor in determining P. Although algorithm 1 does not perform as well as the other two algorithms in minimizing P_B. it still beats them in the overall performance (P). As expected. all P_B values are greater than 0.9. When storage capacity is not so big ($S < 60G$). P_B is close to 1. This is consistent with Theorem 5.1. As more storage is dedicated to replication. λ_B decreases and the total load put on CPU also decreases. This leads to a smaller P_B value.

7.2. Replica Placement in Soft Quality Systems

In this section. we discuss the experiments to evaluate the k-median algorithms: *Greedy*, *Hot Area*. and *Iterative Greedy*. We test the algorithms on a number of synthetic patterns. We vary two factors in generating these patterns: the skewness among quality points and skewness among quality regions. For the first factor. we use the following patterns: uniform. 20/80. 10/90. and Zipf [23]. For the second factor. we use two patterns: scattered and clustered. The clustered pattern is meant to simulate the appearance of hot areas: we artificially create a random number of hot areas by putting some of the hottest replicas into these areas. In the scattered pattern, hot spots are sporadically distributed in the quality space. Fig 5 illustrates various point-level distributions (Zipf. 90/10. 80/20) in scattered and clustered patterns. Unless specified otherwise. we put equal weights on both dimensions thus we calculate utility loss between two quality points by the Manhattan distance between them. Request distributions are generated in two steps: 1. distribute λ to M cells based on the first factor: 2. associate each cell with a quality point based on the second factor.

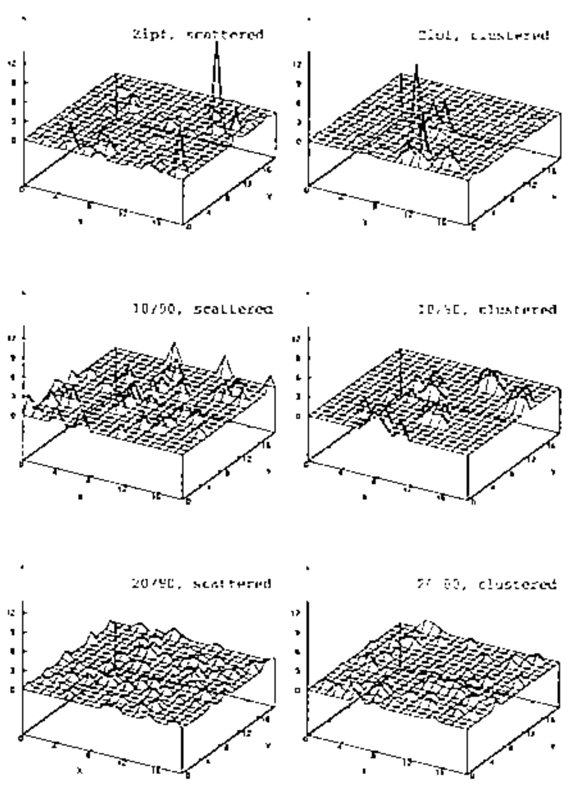

Figure 5. Distributions of request rates in a 2-D quality space.

7.2.1. Utility loss.

We run extensive simulations to test on all combinations of request patterns. We only present the most important results in this section. We first show the utility loss of various algorithms in an experiment with a Zipf distribution (Table 2). The quality space is set to be a small 11×11 grid so that we can compute the optimal replica placement in a brute force way. All data are ratios to corresponding optimal solutions. For comparison, we also added an algorithm named *Random* that randomly chooses replicas and records the best selection in 100 trials. Each point is the average of four experiments with traffic patterns generated using different random seeds. The first thing we can see is that *Iterative Greedy* has the best performance. It actually finds the optimal solution in most cases! The readings of the three other algorithms are at least 7.5% higher than those of *Iterative Greedy*. *Greedy* seems to beat *Hot Area* in the experiments with the scattered distribution but not in the clustered case. The random algorithm performs the worst. Similar results are obtained for other request distribution patterns (data not shown). This is understandable as the chances of getting a good selection of 2 to 4 replicas from 121 are very slim.

To gain more confidence on the initial conclusions drawn from the data in Table 2, we perform similar simulations on a larger scale: we use a 40×40 quality space and test a wider range of K values. Fig 6 shows results for two request distributions (Zipf, 20/80) under the scattered and clustered patterns. This time. all data points are relative utility loss rate

Request Pattern	Algorithm	Replica Number (K)		
		2	3	4
Scattered	*Random*	1.071	1.172	1.367
	Hot Area	1.100	1.121	1.130
	Greedy	1.086	1.093	1.077
	Iterative	1.000	1.000	1.002
Clustered	*Random*	1.213	1.413	1.581
	Hot Area	1.055	1.070	1.168
	Greedy	1.200	1.109	1.090
	Iterative	1.000	1.004	1.011

Table 2. Relative performance of various algorithms to the optimal.

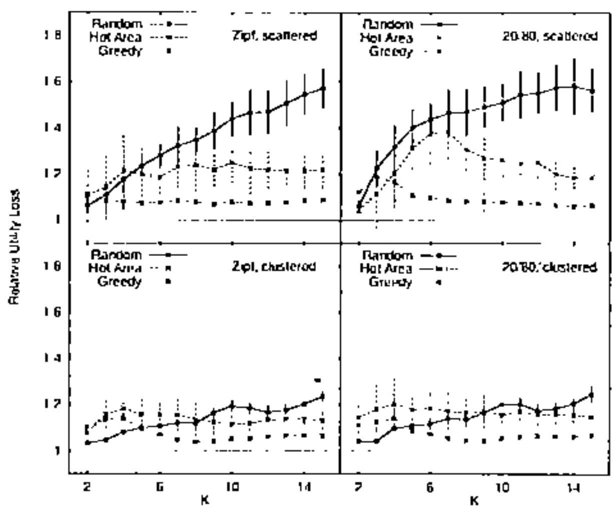

Figure 6. Relative performance of various algorithms to *Iterative Greedy*.

to that of the *Iterative Greedy* and the average of eight experiments (standard deviations are also shown). The superiority of *Iterative Greedy* is once again demonstrated: in all experiments it gets the smallest utility loss. Following *Iterative Greedy* is the *Greedy* algorithm, which is the second best performer in most cases except when K is small. The *Hot Area* algorithm is generally overperformed by *Greedy* and *Random* gives highest utility loss rate in most cases.

According to Fig 6 and Fig 7, only *Random* is sensitive to the increase of K. This is easy to understand; when K increases, the size of the search space also increases. For the other algorithms, increased size of search space does not affect much since they do not act in a blind-search way. Of course, this claim is based on the educated guess that *Iterative Greedy* is insensitive to K as we are comparing everything to it. In this set of experiments, we also compute the optimal values for $K = 2$. On average, there is only a 0.1% difference between the performance of *Iterative greedy* and the optimal value. In about half of the cases, it matches the optimal value. It would be infeasible to compute the optimal solution for $K > 2$ under this situation. It took about 25 minutes to compute a solution for $K = 2$ and a trial on $K = 3$ could last as long as 225 hours!

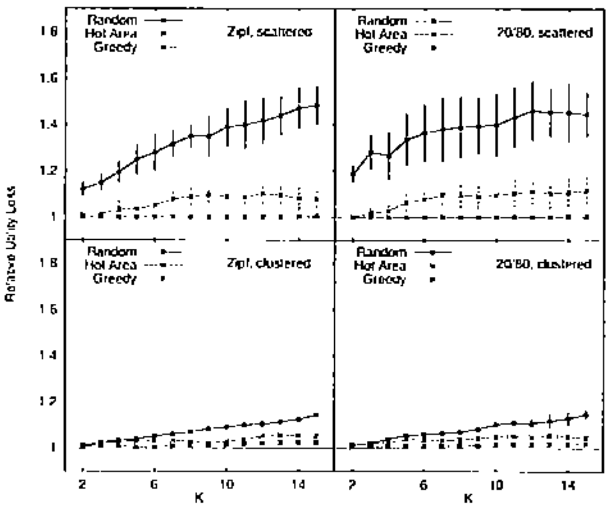

Figure 7. Relative performance to *Iterative Greedy* with type b utility function.

7.2.2. *Hot Area* may be good. Another observation from Fig 6 is that *Hot Area* performs much better in a clustered distribution: only 10% difference from that of *Greedy*. This effect is more clear when we use a utility function that resembles functions *a* and *b* shown in Fig 3B. The feature of this type of function is that utility drops sharply with increased distance to q^{desire}. This means, having replicas placed (very) close to hot areas is the only thing that matters. If we fail to do this, most of the utility in these hot areas will be lost even when there are replicas a few steps away. Fig 7 shows results of the same experiments in Fig 6 except a type *b* utility function is applied. We can clearly see that the performance of *Hot Area* approaches that of *Greedy* and *Iterative Greedy*. In summary, *Hot Area* has good performance in a quality space with highly skewed request distribution. It might be a good choice for making online decisions considering its short running time (Table 3). In these experiments, the advantage of *Iterative Greedy* is marginal. Our interpretation is: by using the extremely skewed distributions and utility functions, the k-median becomes an eas-

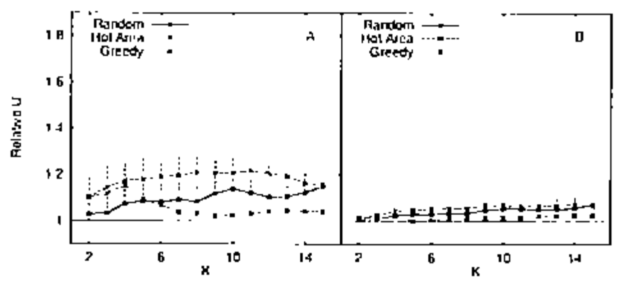

Figure 8. Relative performance of various algorithms to *Iterative Greedy* for uniformly distributed requests. A. type *c* utility function; B. type *b* utility function.

ier problem that can be correctly solved be *Hot Area* and *Greedy*. But that is as far as they can get. What is important is that *Iterative Greedy* is the best in all situations we test.

As mentioned earlier, *Hot Area* only works for skewed distributions of request rates. If this is not the case, the performance of *Hot Area* could be even worse than the *Random* algorithm (Fig 8, which displays the performance under a uniform distribution).

Algorithm	Replica Number (K)			
	2	5	10	15
Random	0.02	0.06	0.11	0.17
Hot Area	0.02	0.03	0.04	0.06
Greedy	2.56	12.88	47.12	102
Iterative + Greedy	9.55	56.26	219.79	493.77
Iterative + Random	6.86	42.68	171.07	385.43
Brute Force	1524	-	-	-

Table 3. Running time of various k-median algorithms.

Table 3 presents the running time of various algorithms under different K values for the above experiments. The experiments are run in a SUN workstation with UltraSparc 1.2MHz CPUs. The results confirm our analysis on the time complexity of these algorithms (Section 6). One exception is that *Random*, which is supposed to have complexity $O(tK)$ (t is the number of random trials and $t = 20$ in our experiments), is actually slower than *Hot Area* at all K values. This is due to the complexity of the random number generator we use. Although *Greedy* is significantly faster than *Iterative Greedy*, the differences are always bounded

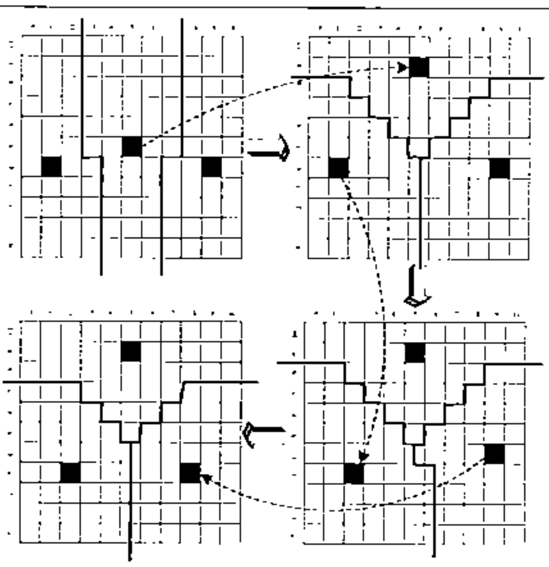

Figure 9. Improvement of replica selection by the Iterative Greedy algorithm.

by a constant: the running time of the latter never exceeds five times that of *Greedy*. One thing to point out is that, in these experiments, we run $2K$ iterations in the *Iterative Greedy* algorithm.

7.2.3. How *Iterative Greedy* works. The above experimental data demonstrated the power of the *Iterative Greedy* algorithm. To get a better idea how it improves replica selection, we monitored the selection process step by step in the above experiments. Fig 9 illustrates a simple yet representative case: in a 11 × 11 2-D quality space, we assign equal weights (λ_k) to each quality point and we use Manhattan distance to calculate utility loss. The four graphs in Fig 9 represent the states of replica selection in order. The algorithm starts by a set of three replicas selected by *Greedy*: $(5,5)$, $(1,4)$, and $(9,4)$, in the order of their being selected. We immediately see that this set of replicas is a bad choice. The reason is: the shape of the regions (divided by thick lines in the figure) the replicas *control* (here A *controls* B means A is the closest replica all members in B can find) are too narrow and long, which gives suboptimal total distances. The ideal shape for a region is one where distances from the replica to all directions are balanced (a circle being the perfect shape in an equally-weighted plane). Also the areas of all regions better be the same. *Iterative Greedy* improves the selection by first moving replica $(5,5)$ to $(5,9)$, creating three regions with better shapes. The movement of $(5,5)$ leaves more points to be controlled by $(1,4)$ on the lower part of the grid so it is moved to $(2,3)$ at the second iteration. Now the two lower regions are unbalanced in their areas (45.5 vs. 37.5). The problem is solved by moving

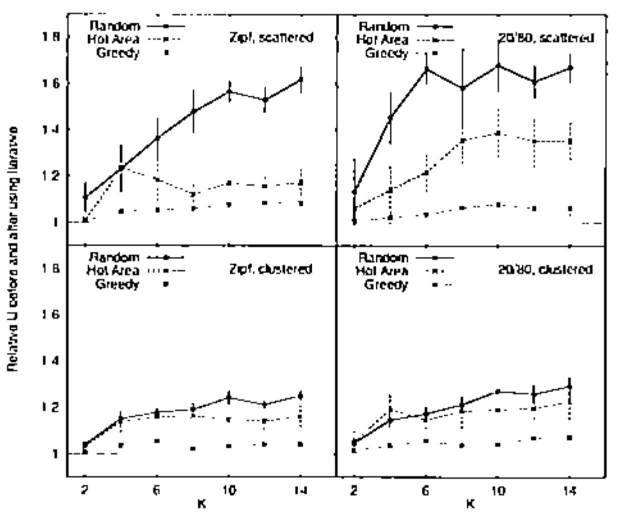

Figure 10. Performance improvement of *Iterative Greedy* to replicas produced by various algorithms.

(4, 9) to (3, 8), which forms an optimal solution. Although further investigations are needed, we somehow get the idea that *Iterative Greedy* works by putting the K replicas one by one to the center of K regions that are equally weighted in the quality space. Similar improvements are also observed in all other scenarios we monitored (data not shown).

7.2.4. More on Iterative Greedy. An important observation is that *Iterative Greedy* works for any initial set of replicas, even one that is chosen randomly. Figure 10 shows the improvement (in terms of relative U) of *Iterative Greedy* to replicas generated by various algorithms. We immediately see the significantly improved performance by running *Iterative Greedy* to these replicas. More interestingly, the extent of improvement (relative U values to those before *Iterative Greedy* is run) is closely related to the performance of the original algorithms. The worse the initial set of replicas, more improvement can we observe. For example, the improvement for *Random* is always higher as the U value of the initial replica list generated is always higher (Fig 6).

On concern is whether the absolute performance of *Iterative Greedy* is affected by the choice of the original replica set. The shape of the curves plotted in Fig 10 resemble those in Fig 6. This implies that similar U values are achieved by *Iterative Greedy* for all initial replica sets. According to Fig 11, the quality of the final replica set chosen is equally good no matter what original algorithm is used to produce the original set. This means that, in practice, we can generate an initial set of replicas using the fastest method such

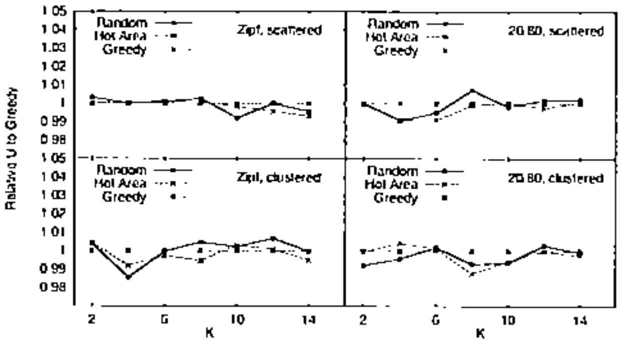

Figure 11. Performance of *Iterative Greedy* using replica sets generated by various algorithms.

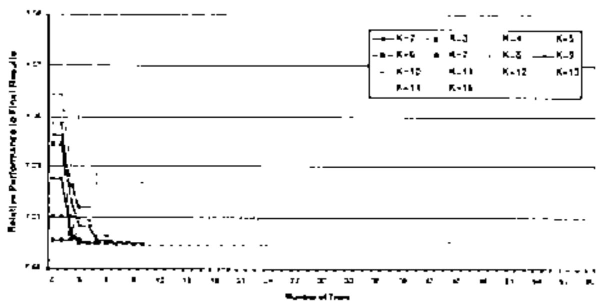

Figure 12. Performance convergence of *Iterative Greedy.*

as *Random* and *Hot Area* without affecting the final performance. This saves the time to run *Greedy* (see Table 3).

Another problem that is related to the time complexity of *Iterative Greedy* is *how fast the U value converges*. We record the U values of intermediate replica sets for up to 100 iterations in all experiments with the 40×40 quality space mentioned in Section 7.2.1. Similar results are obtained for all these experiments thus we only show the result of one in Fig 12. In this experiment, we have an unclustered Zipf distribution and the initial set is generated by *Greedy*. Readings for various K values are plotted in Fig 12. Overally, the results (U) converge fast to a stable value. For most cases, the convergence is complete before the 6-th iteration. The three cases with longer convergence period are those of larger K values: $K = 13$, $K = 14$, and $K = 15$. Even for these cases, convergence is accomplished at the 30-th iteration ($\leq 2K$). At the 15-th iteration, they already reach a U value that is only 1% higher than the final result. There-

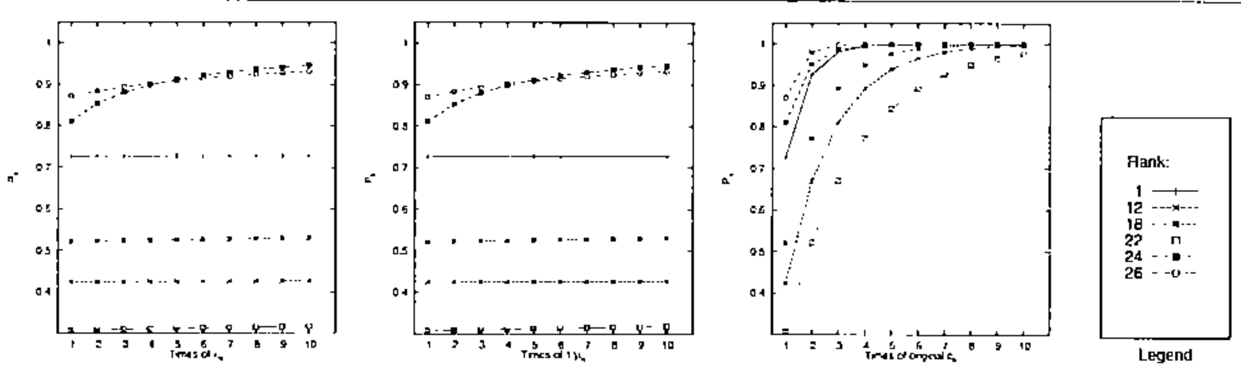

Figure 13. Relationship between traffic class features and blocking probability.

fore, to maintain good performance and short running time, it is sufficient to set the number of iterations to $O(K)$.

8. Conclusions and Future Work

In this paper, we study the placement problem of quality-specific replicas of media data. This problem is generally ignored in multimedia system researches 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 request rate (λ_k) to and storage (s_k) of individual replicas are the most critical factors that affect performance. Other factors such as service rate (μ_k) and resource use (c_k) only play secondary roles. The most interesting discovery in this research is the amazing performance of the *Iterative Greedy* algorithm. It always finds a solution that is close or equivalent to a/the optimal. We also found that the *Hot Area* algorithm could be used in a quality space with clustered request distributions with acceptable performance and short running time.

Our ongoing work in this area include extension of this study to quality-aware data placement in multiple servers and dynamic replication/migration of replicas (i.e. relaxing assumptions in Section 3.1). The optimization towards both utility and blocking rate as well as smart assignment of K to different media objects are all interesting topics to work on. We also think about further exploring the *Iterative Greedy* algorithm from a theoretical point of view.

APPENDIX

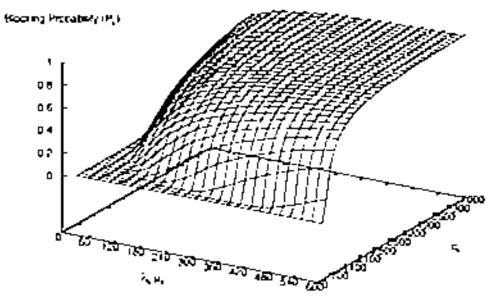

Figure 14. Relationship between the mean of traffic class features and blocking probability.

A. Qualitative Analysis of Equation (3)

In order to study Equation (3), we first need to study the relationship between P_k and the class parameters from Equation (4). It is infeasible to study the qualitative features of Equation (4) analytically under general situations. The effects of traffic class parameters on various performance measures (blocking rate, throughput, etc.) are analyzed for a total of two traffic classes [21]. It is shown that the blocking probability of one class decreases (increases) with the decrease (increase) of μ_k (λ_k) of the other class. This does not give much useful information for our analysis because we are interested in how to choose from all the traffic classes such that P is minimized. What we need here is the qualitative properties of P with respect to the traffic class parameters λ_k, μ_k, and c_k.

We approach this problem by running experiments using a number of synthetic traffic classes with parameters that are close to the scenarios we are interested in. Fig 13 shows the change of P_k with the change of class-specific parameters with a Zipf-like distribution of λ into 27 traffic classes. In each subgraph, the responses of six traffic classes with different ranks (the higher the rank, the larger the λ_k) are displayed. We start by an initial set of class parameters and then change only one parameter of one class (all other parameters remain the same) and record the new P_k value. We can see that P_k increases with the increase (decrease) of λ_k and c_k (μ_k). The change of λ_k and μ_k for lower-ranked classes has less effects on P_k but the latter still monotonically increases. Note the graphs for λ_k and μ_k are identical: a n-time increase of λ_k has the same effect as a n-time decrease of μ_k. This means these two parameters can be viewed as one parameter in the discussion of their effects on P_k. We also test the relationship between P_k and the average λ_k, μ_k, and c_k for all classes and similar results are obtained (Fig 14).

The above leads to the decision to choose classes by their load $\frac{\lambda_k}{\mu_k}c_k$ to minimize P_k. However, we are interested in P instead of P_k and the storage s_k of replicas should also be considered. For the first factor, we believe λ_k has greater effects on P according to Equation (3) (only λ_k appears in it). For the second factor, we simply add it to the selection criteria. Thus, the final selection rule is $\frac{\lambda_k^{\alpha} c_k}{\mu_k s_k}$ ($\alpha > 1$) where α can be generated by further experiments. For the 27 classes discussed, $\alpha = 4$ gives the best results.

B. Theorem 5.1

Proof: Due to the discrete feature of the states, it is very difficult to discuss the characteristics of function (4). Fortunately, Gazdzicki *et al.* [8] gives the following asymptotic approximation to equation (4):

$$P_k = \left(1 - e^{\alpha c_k}\right)\left(1 + o(1)\right) \tag{6}$$

where α is the unique solution to the following equation:

$$\sum_{k=1}^{M} \frac{\lambda_k}{\mu_k} c_k e^{\alpha c_k} = C. \tag{7}$$

Putting (6) into (7), we get

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

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

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

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 (3), we get $P \approx 1$. $\square$

C. The *Greedy* and *Hot Area* Algorithms

```
ADD-REPLICA ($M_i$, slist, k)
1  $U_{min} \leftarrow \infty$
2  for $i \leftarrow 0$ to $M_i - 1$
3      do if $i \notin slist$
4          $u \leftarrow 0$
5          for $j \leftarrow 0$ to $M_i - 1$
6              $u \leftarrow u +$ DISTANCE($j, slist, i$)
7          if $u < U_{min}$
8              $U_{min} \leftarrow u$
9              $slist[k] \leftarrow i$
10 return new utility loss rate of $slist$
```

Greedy calls ADD-REPLICA K times with a *slist* holding k selected replicas so far. It starts with an empty list and $K = 0$. ADD-REPLICA appends a newly selected replica to *slist* and output the total utility loss rate for the new group of selections. It does so by trying all M_i points in the quality space (line 2) to look for the one that yields the smallest utility loss rate. Subroutine DISTANCE gives the smallest utility loss from j to any point in *slist* plus i.

```
HOTAREA ($M_i$, K, size)
1  Set rlist: a list with length $M_i$
2  Set selected: a list with length $K$
3  for $i \leftarrow 0$ to $M_i$
4      do $rlist[ind].id \leftarrow i$
5          $rlist[ind].rate \leftarrow$ RATE-SUM($i, size$)
6  sort rlist by decreasing order of rate
7  $selected[0] \leftarrow$ BEST-POINT($rlist[0].id, size$)
8  $k \leftarrow 1$
9  for $i \leftarrow 1$ to $M_i$
10     if OVERLAP($rlist[i].id, selected$) is FALSE
11         $selected[k] \leftarrow$ BEST-POINT($rlist[i].id, size$)
12         $k \leftarrow k + 1$
13         break loop if $k$ reaches $K$
14 return selected
```

The procedure HOTAREA first calculates the accumulated request rate in all virtual cubes with side length *size* and sorts them by such rate (line 1 to line 6). The K replicas are then assigned to the hottest cubes. One important thing is to eliminate cubes that partially overlap with the ones that are already selected (line 10). The subroutine OVERLAP checks whether a cube overlaps with any selected cubes in *selected*. BEST-POINT gives the best location to place a replica within a cube with side length *size*.

References

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

[2] AvantGo:, http://www.avantgo.com.

[3] E. Bertino, A. Elmagarmid, and M.-S. 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] I. Foster, A. Roy, and V. Sander. A Quality of Service Architecture that Combines Resources Reservation and Application Adaptation. In *Proceedings of IWQOS*, pages 181–188, June 2000.

[7] A. Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer. Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives. *IEEE Personal Communications*, 5(4), 1998.

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

[9] 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.

[10] 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.

[11] 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, December 1979.

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

[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] W. Li. Overview of Fine Granularity Scalability in MPEG-4 Video Standard. *IEEE Trans. Circuits and Systems for Video Technology*, 11(3):301–317, Mar. 2001.

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

[16] 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.

[17] T. Loukopoulos, I. Ahmad, and D. Papadias. An Overview of Data Replication on the Internet. In *Proceedings of International Symposium of Parallel Architectures, Algorithms and Networks*, pages 694–711, 2002.

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

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

[20] R. Mohan, J. R. Smith, and C.-S. Li. Adapting Multimedia Internet Content for Universal Access. *IEEE Transactions on Multimedia*, 1(1):104–114, March 1999.

[21] P. Nain. Qualitative Properties of the Erlang Blocking Model with Heterogeneous User Requirements. *Queueing Systems: Theory and Applications*, 6(2):189–206, April 1990.

[22] S. Nepal and U. Srinivasan. DAVE: A System for Quality Driven Adaptive Video Delivery. In *Proceedings of Intl. Workshop of Multimedia Information Retrieval (MIR)*, pages 224–230, November 2003.

[23] 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.

[24] G. On, J. Schmitt, M. Liepert, and R. Steinmetz. Replication with QoS support for a Distributed Multimedia System. In *Proceedings of the 27th EUROMICRO Conference (Workshop on Multimedia and Telecommunications), Warszaw, Poland*, IEEE, Sept. 2001. ISBN 0-7695-1236-4.

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

[26] S. Prabhakar, D. Agrawal, A. E. Abbadi, A. Singht, and T. Smith. Browsing and Placement of Multi-Resolution Images on Parallel Disks. *Springer/ACM Multimedia Systems*, 8(6):459–469, January 2003.

[27] 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.

[28] 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. Herakolin, Crete, Greece.

[29] 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 of *IFIP Conference Proceedings*. Kluwer, 1998.

[30] 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.

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