Quality-Aware Replication of Multimedia Data

Yi-Cheng Tu, Jingfeng Yan, and Sunil Prabhakar

Purdue University, West Lafayette, IN 47906, U.S.A.

Abstract. In contrast to alpha-numerical data, 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 applications, or the limitations of their resources. 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 a soft quality system model where users are willing to negotiate their quality needs. An important optimization goal under such a model is to minimize utility loss. We propose a powerful greedy algorithm to solve this optimization problem. Extensive simulations show that our algorithm finds near-optimal solutions. The algorithm is flexible in that it can be extended to deal with replica selection for multiple media objects and changes of query pattern. We also discuss an extended version of the algorithm with potentially better performance.

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. Users can request data with specific quality requirements due to the needs of their applications, or the limitations of their resources. Quality-aware multimedia systems [1,2,3] allow users to specify the *quality* of the media to be delivered. The quality parameters of interest also differ by the type of media that we deal with. For digital videos, which we use as example throughout this paper, the quality parameters include resolution, frame rate, color depth, audio quality, compression format, security level, and so on [4]. For example, a video editor may request a video at very high resolution when editing it on a high-powered workstation, but request the video at low resolution and frame rate when viewing it using a PDA.

Generally, there are two approaches to satisfy user quality specifications: (i) *dynamic adaptation*: store only the highest resolution copy, and convert it to the quality format requested by the user as needed at run-time; or (ii) *static*

K.V. Andersen, J. Debenham, and R. Wagner (Eds.): DEXA 2005, LNCS 3588, pp. 240–249, 2005.

adaptation: pre-compute each different quality that can be requested and store them on disk. When a user query is received, the appropriate copy is retrieved from disk and sent to the user. Dynamic adaptation suffers from a very high CPU overhead for transcoding from one quality to another [5,6]. Therefore, real-time adaptation is difficult in a multi-user environment. *Static adaptation* attempts to solve the problem of high CPU cost 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.

However, storage costs for static adaptation could be extremely high. This is because users vary widely in their quality needs and resource availability [5]. This leads to a large number of quality-specific copies of the same media content that need to be stored on disk. From a service provider's point of view, the storage requirements for static adaptation should not grow unboundedly as storage, although cheap, is not free. This is especially true for commercial media databases that must provide high reliability of disk resources. Our analysis [7] shows that the extra disk space needed to accommodate all possible qualities is $O(n^d)$ times of the original copy where n is the number of quality levels in one quality dimension and d is the number of quality dimensions. Therefore, it is infeasible to store all possible quality copies. On the other hand, the strategy of selecting few copies based on the bandwidth of user devices (T1, DSL, dial-up ...), as many media services do nowadays, ignores the diversity of user's quality needs. In this paper, we study the problem of quality selection under storage constraints for the purpose of satisfying user quality requirements.

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 [8]. Quality-aware replication 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 offline transcoding. In this paper, we assume user behavior can be described by a *soft quality* model where users are willing to negotiate when the original quality required is not available. Under this situation, users may accept a different quality with a decrease of satisfaction with the service. Our data replication algorithms are designed to achieve the highest user satisfaction under fixed resource (storage) capacities. Quality selection under a *hard quality* model where users have rigid quality requirements is discussed in our technical report [7]. There are two major contributions of this paper:

1. We formulate the replica selection problem as an optimization with the goal of maximizing user satisfaction. We propose a fast greedy algorithm with comparable performance to commercial optimizers. An improvement to the greedy algorithm is also discussed.
2. We extend the above algorithm to handle the situation of *dynamic replication* where changes of query pattern are expected. Our solution is fast and achieves the same level of optimality as the original algorithm.

2 Related Work

Quality adaptation in media delivery in response to heterogeneous client requests has attracted a lot of attention [2,6]. However, quality selection in static adaptation is not well addressed. A closely related work is [5] where quality selection (under storage constraint) is performed to achieve the smallest transcoding costs. In [9], the problem of optimal materialized view selection is studied. Both [5] and [9] address different data selection problems from ours. Furthermore, neither considers quality selection in response to dynamic changes of query pattern.

Efforts to build quality-aware media systems include [2,6,3,1]. In our previous work [1], we extend the query generation/optimization module of a multimedia DBMS to handle quality of queries as a core DBMS functionality. In [4], specification of quality parameters in multimedia databases is discussed. The traditional data replication problem has been studied extensively in the context of web [10,11], distributed databases [8], and multimedia systems [12,13]. 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 [13,14]. Unlike web and multimedia data, database contents are accessed by both read and write operations. This leads to high requirements for data consistency, which often conflict with data availability. Another important issue is dynamic data replication. As access rates to individual data items are likely to change, we need to make our replication strategy adapt to changes quickly and accurately to achieve optimal long-term performance. Wolfson *et al.* [15] introduced an algorithm that changes the location of replicas in response to changes of read-write patterns of data items.

3 System Model and Problem Statement

We assume that the database consists of a collection of servers that host the media content and service user queries. Servers have limited storage space S. For now, we consider only one media object and in Section 4 we extend our discussions to a system with V media objects. User requests identify (via a query) an object to be retrieved as well as the desired quality requirements on d quality dimensions. Each quality can thus be modeled as a point in a d-dimensional quality space. The domain of a quality parameter consists of finite number of values and we denote the total number of quality points as m. Each possible quality k is modeled by f_k and s_k where f_k represents the query rate for this version of the media and s_k is the byte size of this replica.

Utility is frequently used to quantify user satisfaction on a service [16] and is thus the primary optimization goal in quality-critical applications [17]. *Utility functions* serve the purpose of mapping quality to utility. For a request to a quality A, if A is replicated, the server retrieve that replica to serve the request

and no utility is lost. Otherwise, the request is served by the closest replica B to A, and utility loss increases with the distance between A and B. Note that providing a higher quality than needed does cause utility loss because the client device may not have enough resources to handle it [7]. Techniques for generating utility functions can be found in [16].

Our problem is to pick a set L of replicas from the quality space 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 quality points and $u(j, L)$ is the utility with which quality j is served by the closest replica in L. We set $u(j, L)$ to be an decreasing function (within the range of $[0, 1]$) of the distance between j and its nearest neighbor in L (see [7] for more details) and we have $u(j, L) = 1.0$ if $j \in L$. We weight the utility by the request rate f_j and the weighted utility is termed as *utility rate*. We name our problem the *fixed-storage replica selection* (FSRS) problem and it can be formulated as the following integer program:

$$\begin{aligned}
\text{maximize} \quad & \textstyle\sum_{j \in J} \sum_{k \in J} f_j u(j,k) Y_{jk}, && (1)\\
\text{subject to} \quad & \textstyle\sum_{k \in J} X_k s_k \leq S, && (2)\\
& \textstyle\sum_{k \in J} Y_{jk} = 1, && (3)\\
& Y_{jk} \leq X_k, && (4)\\
& Y_{jk} \in \{0, 1\}, && (5)\\
& X_k \in \{0, 1\}. && (6)
\end{aligned}$$

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 (2) shows the storage constraint while Equations (3) and (4) mean that 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.

A close match to FSRS is the so-called p-median problem with the same problem statements except Equation (2) becomes $\sum X_k = p$, meaning only p ($p < |J|$) points are to be selected. As the p-median problem is NP-hard [18], it is easy to see that FSRS is also NP-hard [7].

4 Replica Selection Algorithms

In dealing with the FSRS problem, we can use a benefit/cost model to analyze the value of a replica k: the cost is obviously the storage s_k, the benefit is the gain of utility rate by selecting k. A good heuristic would select the set of replicas with the highest benefit/cost ratios [7]. However, the benefit of one replica depends on the selection of other replicas. We propose an algorithm (Fig 1) that takes greedy 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 use $\Delta U_k / s_k$ to denote 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, i.e. the n-th replica maximizes $\Delta U_n / s_n$ based on the $n - 1$ replicas that are already selected.

```
Algorithm GREEDY
Inputs: f_k, s_k, S
Output: a set of selected replicas, P

1  s' ← S, P ← ∅, k ← 0
2  while k ≠ NULL do
3        k ← ADDREPLICA(s', P)
4        s' ← s' − s_k
5        append k to P
6  return P
```

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

Fig. 1. The *Greedy* algorithm

Algorithm *Greedy* (Fig 1) terminates when no more replicas can be added due to storage constraints. New replicas are selected via subroutine AddReplica by trying all m points in the quality space to look for the one that yields the largest utility density. Note subroutine MaxUtil (line 6 of AddReplica) gives the utility from j to its nearest replica in $P' + k$, which can be done in constant time. As AddReplica runs for $O(m^2)$ time, the time complexity for *Greedy* is $O(Im^2)$ where I is the total number of replicas selected.

The Iterative Greedy Algorithm. This algorithm attempts to improve the performance of *Greedy*. We notice that at each step of *Greedy*, some local optimization is achieved: the $(n+1)$-th replica chosen *is* the best given the first n replicas. The problem is: we do not know if the first n replicas are good choices. However, we believe the $(n+1)$-th replica added is more 'reliable' than its predecessors because more global information (existence of other selected replicas) is leveraged in its selection. Based on this conjecture, we develop the *Iterative Greedy* algorithm that iteratively improves the 'correctness' of the replicas chosen. Specifically, we repeatedly get rid of the most 'unreliable' selected replica and choose a new one. The operations in *Iterative Greedy* are shown in Fig 2. All replicas selected by *Greedy* are stored in a FIFO queue P'. In each iteration, we dequeue P' and find one replica (again, by AddReplica) based on the remaining replicas. The newly identified replica is then added to the tail of P'. We record the set of replicas with the largest utility rate as the final output (P). 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 the number of replicas selected by *Greedy*. The time complexity of *Iterative Greedy* is thus $O(Im^2)$, which is the same as that of *Greedy*.

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 media. The different content in the physical media can be modeled as a new quality

```
Algorithm IterativeGreedy
1  U_max ← 0, P ← ∅
2  for i ← 0 to I
3      do k ← dequeue P'
4         s' ← s' + s_k
5         l ← AddReplica(s', P')
6         append l to P' and update s'
7         U ← total utility rate of P'
8         if U_max < U then
9           do U_max ← U
10             copy P' to P
```

Fig. 2. The *Iterative greedy* algorithm. **Output:** P. **Inputs:** P' - a set of replicas selected, s' - available storage after P' is replicated.

dimension called *content*. A special feature of *content* is its lack of adaptability. For example, any replica of the movie *Matrix* cannot be used to serve a request to the movie *Shrek*. Assume the quality spaces of all physical media have m points, the FSRS problem with V media can thus be solved by simply running the *Greedy* algorithm for the virtual media with Vm 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 AddReplica for those with the same content. Thus, the time complexity of Greedy is $O(IVm^2)$ rather than $O(IV^2m^2)$.

5 Dynamic Data Replication

In previous sections we deal with the problem of *static* replication, in which access rates of all qualities do not change over time. In this section, we discuss quality-aware data replication in an environment where access patterns change. A good dynamic replication algorithm needs to meet two requirements: quick response to changes and optimality of results. 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 choices 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. In this section, we assume that runtime changes 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. This assumption is found to be reasonable in many systems [12,13]. We understand that a solution for more general situation is also meaningful and we leave it as future work.

Let us first investigate how algorithm Greedy selects replicas. The history of total utility rate gained and storage spent on each selected replica can be represented as a series of points in a 2D graph. We call the lines that connect these points in the order of their being selected a *Replication Roadmap* (RR). Fig 3 shows two examples of RRs plotted with the same scale. We can see that all replication roadmaps are convex. This is because: the slope of the line connecting any two consecutive points (e.g. $r1$ and $r2$ in Fig 3A) represents the ratio of ΔU_{r2} to s_{r2}. As *Greedy* always chooses a replica with the largest $\Delta U/s$ value, the slopes of the line segments along the RR are thus non-increasing.

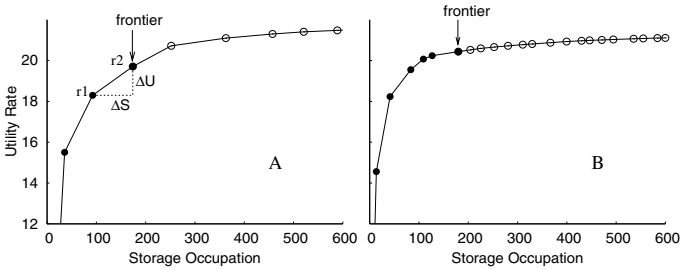

Fig. 3. Replication roadmaps

In dynamic replication, replicas need to be re-selected with respect to the new query rate of a media object. Suppose the query rate f_i of a media object i increases by a factor δ ($\delta > 0$). 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 media object whose slope in the last piece of RR is small. Take Fig 3 as an example. Suppose we have fully extended RRs: all future replicas are precomputed (empty dots in Fig 3) 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: 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 design an online algorithm named SoftDynaRep for dynamic replication (see [7] for pseudocode).

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 replicas. For all RRs, we put the immediate predecessor of the *frontier* in a list called *blist* and its immediate successor in a list called *flist*. Both lists are sorted by the slopes of the segments stored. The *Preprocess phase* runs at $O(Vm^3)$ time and it only needs to be executed once. The *Online Phase* is triggered once a change in query rate to an object i is detected. The idea is to iteratively take storage from the tail of *blist* and add that to the head of *flist* (we call this operation *storage exchange*) until a new equilibrium is reached. The running time of this phase is $O(I_e \log V)$ where I_e is the number of storage exchanges (obviously, $I_e = O(m)$). We claim that the online phase of SoftDynaRep achieves the same quality in the selected replicas as that by rerunning *Greedy*. A rigorous proof can be found in [7].

6 Experiments

We study the behavior of the proposed algorithms by extensive simulations. Due to space limitations, we only present the most important results. We use traces of 270 MPEG-1 videos extracted from a real video database[1] as experimental data. We simulate a 3D quality space with various number (100-500) of replicas. The s_k values for all replicas are generated from empirical equations [7]. As real-world

[1] http://www.cs.purdue.edu/vdbms

traffic traces for quality-aware systems are not available, we test various synthetic access patterns (i.e., f_k values) in our simulations. We run our experiments on a Sun Workstation with a UltraSparc 1.2GHz CPU.

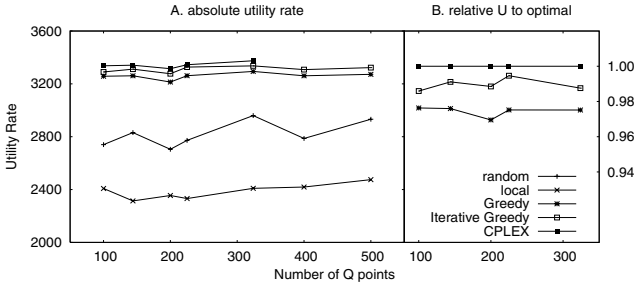

Fig. 4. Optimality of replica selection algorithms. Each data point represents the mean of four experiments.

Performance of Replica Selection Algorithms. We evaluate the performance of *Greedy* and *Iterative Greedy* algorithms in terms of optimality (Fig 4) and running time (Fig 5). 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 CPLEX mathematical programming package[2]; 2. a random algorithm; 3. a *local* algorithm that places replicas in the most frequently accessed 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 4A, it is clear that our algorithms always find solutions that are very close to the optimal. More details can be found in Fig 4B 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 least half in all cases: the solutions it finds always achieve more than 99% of the optimal utility rate. For both algorithms, the performance is insensitive to the number of quality points. Nor is it affected by access patterns or storage constraints. We tested different access patterns (e.g. Zipf, 20-80, and uniform) and S values (60-300GB) and obtained similar results (data not plotted here). The solutions given by *random* and *local* are far from optimal. The fact that the *local* algorithm performs even worse than the random algorithm 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 Figure 5. CPLEX is the slowest algorithm in all cases. This is what we expected as its target is the global optimal solution. 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.

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

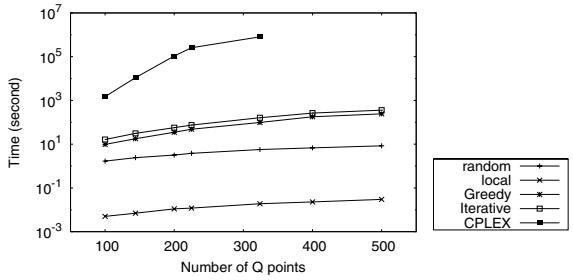

Fig. 5. Running time of different replica selection algorithms

Performance of dynamic replication algorithm. We also test our dynamic replication algorithm under 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*. For all events, the replicas selected by SoftDynaRep get utility rates that are consistently within 99.5% of that by rerunning the *Greedy* algorithm (Fig 6A). 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 Greedy 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 6B, we record such numbers for each execution of SoftDynaRep and very few of these readings exceed 15.

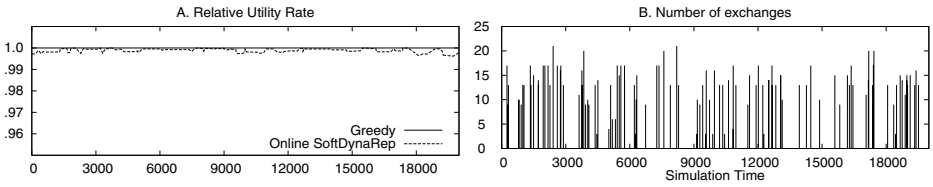

Fig. 6. Performance of SoftDynaRep

7 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 oversimplified assumption that storage space is always abundant. We provide solutions to the problem under a soft quality model where users' quality needs are negotiable. We propose a greedy algorithm to solve the optimal replica selection problem heuristically. Experiments show that the total utility rates of the solutions found by our algorithm are over 97% of those of the optimal. An advanced version of this algorithm further increases that to 99%. A derived online algorithm provides an elegant solution to an important subproblem of dynamic data replication. While the greedy algorithm takes cubic time, the low time complexity of the online algorithm makes our solution scalable.

References

1. Tu, Y.C., Prabhakar, S., Elmagarmid, A., Sion, R.: QuaSAQ: An Approach to Enabling End-to-End QoS for Multimedia Databases. In: Proceedings of EDBT. (2004) 694–711
2. Nepal, S., Srinivasan, U.: DAVE: A System for Quality Driven Adaptive Video Delivery. (In: Proceedings of Intl. Workshop of Multimedia Information Retrieval (MIR04)) 224–230
3. HAfiD, A., Bochmann, G.: An Approach to Quality of Service Management in Distributed Multimedia Application: Design and Implementation. Multimedia Tools and Applications **9** (1999) 167–191
4. Bertino, E., Elmagarmid, A., Hacid, M.S.: A Database Approach to Quality of Service Specification in Video Databases. SIGMOD Record **32** (2003) 35–40
5. Lum, W.Y., Lau, F.C.M.: On Balancing Between Transcoding Overhead and Spatial Consumption in Content Adaptation. In: Proceedings of MOBICOM. (2002) 239–250
6. Mohan, R., Smith, J.R., Li, C.S.: Adapting Multimedia Internet Content for Universal Access. IEEE Transactions on Multimedia (**1**) 104–114
7. Tu, Y.C., Prabhakar, S.: Quality-Aware Replication of Multimedia Data. Technical Report CSD-TR-0423, Purdue University (2004)
8. Nicola, M., Jarke, M.: Performance Modeling of Distributed and Replicated Databases. IEEE Trans. Knowledge and Data Engineering **12** (2000) 645–672
9. Harinarayan, V., Rajaraman, A., Ullman, J.D.: Implementing Data Cubes Efficiently. In: Proceedings of SIGMOD. (1996) 205–216
10. Rabinovich, M.: Issues in Web Content Replication. Data Engineering Bulletin **21** (1998) 21–29
11. Qiu, L., Padmanabhan, V.N., Voelker, G.M.: On the Placement of Web Server Replicas. In: Proceedings of IEEE INFOCOM. (2001) 1587–1596
12. Little, T.D.C., Venkatesh, D.: Popularity-Based Assignment of Movies to Storage Devices in a Video-on-Demand System. Springer/ACM Multimedia Systems **2** (1995) 280–287
13. Wang, Y., Liu, J.C.L., Du, D.H.C., Hsieh, J.: Efficient Video File Allocation Schemes for Video-on-Demand Services. Springer/ACM Multimedia Systems **5** (1997) 282–296
14. Dan, A., Sitaram, D.: An Online Video Placement Policy Based on Bandwidth to Space (BSR). In: Proceedings of ACM SIGMOD. (1995) 376–385
15. Wolfson, O., Jajodia, S., Huang, Y.: An Adaptive Data Replication Algorithm. ACM Transactions on Database Systems **22** (1997) 255–314
16. Menges, G.: 2. In: Economic Decision Making: Basic Concepts and Models. Longman (1973) 21–48
17. Lee, C., Lehoczky, J., Siewiorek, D., Rajkumar, R., Hansen, J.: A Scalable Solution to the Multi-Resource QoS Problem. In: Proceedings of the IEEE Real-Time Systems Symposium. (1999)
18. Kariv, O., Hakimi, S.L.: An Algorithmic Approach to Network Location Problems. II: The p-Medians. SIAM Journal of Applied Mathematics (**37**) 539–560