Note that, by the nature of EPPV service, different display phases of the same clip will typically (assuming the clip's period is smaller than its length) overlap in time, and, consequently, share resources such as disk bandwidth and buffer space. This presents possibilities for additional optimizations for VBR data under EPPV, by controlling the overlap of concurrent streams on the same clip to reduce their combined resource requirements (e.g., overlapping bandwidth "peaks" in one phase with "valleys" in another). Exploring such optimizations is left as an open issue for future research.

References

[1] E.L. Abram-Profeta and K.G. Shin. "Scheduling Video Programs in Near Video-on-Demand Systems". In *Proc. of ACM Multimedia '97*, pp. 359–369, Seattle, Washington, November 1997.

[2] C.C. Aggarwal, J.L. Wolf, and P.S. Yu. "On Optimal Batching Policies for Video-on-Demand Storage Servers". In *Proc. of the 1996 Intl. Conf. on Multimedia Computing and Systems*, pp. 253–258, Hiroshima, Japan, June 1996.

[3] C.C. Aggarwal, J.L. Wolf, and P.S. Yu. "The Maximum Factor Queue Length Batching Scheme for Video-on-Demand Systems". Technical Report RC 20621 (11/11/96), IBM Research Division, 1996.

[4] K.C. Almeroth and M.H. Ammar. "On the Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service". *IEEE Journal on Selected Areas in Communications*, August 1996.

[5] S. Baruah, L. Rosier, I. Tulchinsky, and D. Varvel. "The Complexity of Periodic Maintenance". In *Proc. of the 1990 Intl. Computer Symposium*, pp. 315–320, Taiwan, 1990.

[6] M.Y. Chan and F. Chin. "Schedulers for Larger Classes of Pinwheel Instances". *Algorithmica*, 9:425–462, 1993.

[7] M.-S. Chen, D.D. Kandlur, and P.S. Yu. "Optimization of the Grouped Sweeping Scheduling (GSS) with Heterogeneous Multimedia Streams". In *Proc. of ACM Multimedia '93*, pp. 235–242, Anaheim, California, August 1993.

[8] A. Dan, D. Sitaram, and P. Shahabuddin. "Scheduling Policies for an On-Demand Video Server with Batching". In *Proc. of ACM Multimedia '94*, pp. 15–23, San Francisco, California, October 1994.

[9] M.R. Garey and D.S. Johnson. *"Computers and Intractability: A Guide to the Theory of NP-Completeness"*. W.H. Freeman, 1979.

[10] M.N. Garofalakis, B. Özden, and A. Silberschatz. "Resource Scheduling in Enhanced Pay-Per-View Continuous Media Databases". In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pp. 516–525, Athens, Greece, August 1997.

[11] D.J. Gemmell, H.M. Vin, D.D. Kandlur, P.V. Rangan, and L.A. Rowe. "Multimedia Storage Servers: A Tutorial". *IEEE Computer*, 28(5):40–49, May 1995.

[12] L. Golubchik, J.C.S. Lui, and R.R. Muntz. "Adaptive Piggybacking: A Novel Technique for Data Sharing in Video-on-Demand Storage Servers". *ACM Multimedia Systems*, 4(3):140–155, 1996.

[13] C.-C. Han, C.-J. Hou, and K.-J. Lin. "Distance-Constrained Scheduling and Its Applications to Real-Time Systems". *IEEE Trans. on Computers*, 45(7):814–826, July 1996.

[14] O.H. Ibarra and C.E. Kim. "Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems". *Journal of the ACM*, 22(4):463–468, October 1975.

[15] M. Kamath, K. Ramamritham, and D. Towsley. "Continuous Media Sharing in Multimedia Database Systems". In *Proc. of the 4th Intl. Conf. on Database Systems for Advanced Applications (DASFAA'95)*, pp. 79–86, Singapore, April 1995.

[16] D.E. Knuth. *"The Art of Computer Programming (Vol. 2 / Seminumerical Algorithms)"*. Reading, Mass. : Addison-Wesley Pub. Co., 1981.

[17] E.L. Lawler. "Fast Approximation Algorithms for Knapsack Problems". *Mathematics of Operations Research*, 4(4):339–356, November 1979.

[18] M.Y.Y. Leung, J.C.S. Lui, and L. Golubchik. "Buffer and I/O Resource Pre-allocation for Implementing Batching and Buffering Techniques for Video-on-Demand Systems". In *Proc. of the 13th Intl. Conf. on Data Engineering*, Birmingham, U.K., April 1997.

[19] T.D.C. Little and D. Venkatesh. "Popularity-Based Assignment of Movies to Storage Devices in a Video-on-Demand System". *ACM Multimedia Systems*, 2:280–287, 1995.

[20] C.L. Liu and J.W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". *Journal of the ACM*, 20(1):46–61, January 1973.

[21] C. Martin, P.S. Narayanan, B. Özden, R. Rastogi, and A. Silberschatz. "The $\mathcal{F}ellini$ Multimedia Storage Server". In *"Multimedia Information Storage and Management", S.M. Chung (Ed.)*. Kluwer Academic Publishers, 1996.

[22] B. Özden, A. Biliris, R. Rastogi, and A. Silberschatz. "A Low-Cost Storage Server for Movie on Demand Databases". In *Proc. of the 20th Intl. Conf. on Very Large Data Bases*, pp. 594–605, Santiago, Chile, September 1994.

[23] B. Özden, A. Biliris, R. Rastogi, and A. Silberschatz. "A Disk-Based Storage Architecture for Movie on Demand Servers". *Information Systems*, 20(6):465–482, 1995.

[24] B. Özden, R. Rastogi, and A. Silberschatz. "A Framework for the Storage and Retrieval of Continuous Media Data". In *Proc. of the 1995 Intl. Conf. on Multimedia Computing and Systems*, pp. 2–13, Washington, D.C., May 1995.

[25] B. Özden, R. Rastogi, and A. Silberschatz. "Disk Striping in Video Server Environments". In *Proc. of the 1996 Intl. Conf. on Multimedia Computing and Systems*, Hiroshima, Japan, June 1996.

[26] B. Özden, R. Rastogi, and A. Silberschatz. "On the Design of a Low-Cost Video-on-Demand Storage System". *ACM Multimedia Systems*, 4(1):40–54, 1996.

[27] B. Özden, R. Rastogi, and A. Silberschatz. "The Storage and Retrieval of Continuous Media Data". In *"Multimedia Database Systems: Issues and Research Directions", V.S. Subrahmanian and Sushil Jajodia (Eds.)*, pp. 237–261. Springer-Verlag, 1996.

[28] B. Özden, R. Rastogi, and A. Silberschatz. "Periodic Retrieval of Videos from Disk Arrays". In *Proc. of the 13th Intl. Conf. on Data Engineering*, Birmingham, U.K., April 1997.

[29] B. Özden, R. Rastogi, A. Silberschatz, and C. Martin. "Demand Paging for Movie-on-Demand Servers". In *Proc. of the 1995 Intl. Conf. on Multimedia Computing and Systems*, Washington, D.C., May 1995.

[30] D.A. Patterson, G.A. Gibson, and R.H. Katz. "A Case for Redundant Arrays of Inexpensive Disks (RAID)". In *Proc. of the 1988 ACM SIGMOD Intl. Conf. on Management of Data*, pp. 109–116, Chicago, Illinois, June 1988.

[31] PRECEPT Software, Inc. IP/TV Datasheets. (`http://www.precept.com/datasheets/html/iptvds1.htm`).

[32] P.V. Rangan and H.M. Vin. "Efficient Storage Techniques for Digital Continuous Multimedia". *IEEE Trans. on Knowledge and Data Engineering*, 5(4):564–573, August 1993.

[33] S. Sahni. "Approximate Algorithms for the 0/1 Knapsack Problem". *Journal of the ACM*, 22(1):115–124, January 1975.

[34] H. Shachnai and P.S. Yu. "The Role of Wait Tolerance in Effective Batching: A Paradigm for Multimedia Scheduling Schemes". Technical Report RC 20038 (88607), IBM Research Division, April 1995.

[35] W. Shi and S. Ghandeharizadeh. "Buffer Sharing in Video-On-Demand Servers". *ACM SIGMETRICS Bulletin*, 25(2):13–20, September 1997.

[36] A. Silberschatz and P. Galvin. *"Operating System Concepts"*. Addison-Wesley Publishing Company, 1998. (Fifth Edition).

[37] J.A. Stankovic and K. Ramamritham, editors. *"Advances in Real-Time Systems"*. IEEE Computer Society Press, Los Alamitos, California, 1993.

[38] W.D. Wei and C.L. Liu. "On a Periodic Maintenance Problem". *Operations Research Letters*, 2(2):90–93, June 1983.

combination "Coarse-grained Striping + Scheduling Trees" emerged as the clear winner, exhibiting excellent scaleup under all workload scenarios. Again, for a detailed discussion, the reader is referred to [10].

4 Open Problems and Research Directions

The EPPV service model and the scheduling tree framework outlined in this paper suggest several directions for future research.

(1) *Handling Task Release/Departure Dates.* The schemes presented in this paper are based on the assumption that the periodic tasks corresponding to the retrieval of clips from the EPPV server are all available to be scheduled at time 0. Furthermore, the tasks are *permanent*, in the sense that, once scheduled, they will not depart from the system (at least, not until the server decides to reschedule its offerings). In real life, however, the clip retrievals will have specific *release* and *departure* dates that limit their lifetime in the playback program. For example, the offering of an R-rated presentation should probably be limited to the evening hours (e.g., 9PM to 5AM). A possible solution using the existing methods, would be to construct multiple scheduling trees for different time intervals (based on expected releases/departures) and then constrain the algorithm that combines trees to only combine trees for the same interval. A more general approach would be to try to directly incorporate the concept of release and departure dates in the scheduling tree structure. The main idea is that the scheduling structure needs some way of accounting for *temporary* periodic tasks that are active only during specific time intervals. This essentially corresponds to an extension of our model of periodic real-time tasks where the *offering of a clip itself* can be either periodic (e.g., daily 9PM–5AM) or one-shot (e.g., a live debate on November 12). Designing scheduling structures and algorithms for this extended task model is a challenging and important problem for future research.

(2) *Incorporating Memory Buffering.* The schemes presented in this paper assume that: (1) the minimum possible amount of buffering is provided to each active stream; and, (2) the buffering constraint can never be the bottleneck when deciding the subset of the clips to schedule (i.e., there will always be enough buffers to support the clips chosen). When server memory is scarce, assumption (2) may not be valid. In such cases, the buffering constraint can be incorporated in our scheduling framework by viewing buffer requirements as an additional dimension to the "size" of a clip. Our schemes can then be readily extended using a *vector packing* formulation, as in the 2-dimensional PACKCLIPS algorithm[9] . Note that the buffer constraint can be ignored if the buffer size is at least $2 \cdot T \cdot r_{disk} \cdot n_{disk}$. The situation is much more interesting when assumption (1) is invalid. The existence of additional memory buffers allows the clip retrieval tasks some flexibility, in the sense that data for a stream can be retrieved within a range of several rounds (rather than the round right before its consumption). This flexibility suggests connections to traditional real-time scheduling problems, like the *pinwheel problem* [6] or *distance-constrained task scheduling* [13]. If this additional buffer memory is shared among streams, then a number of interesting issues arise with respect to buffer management policies.

(3) *Better Heuristics for Building Scheduling Trees.* Although our experiments have shown that the greedy BUILDTREE heuristic performs very well in practice, we have not been able to obtain theoretical bounds on its worst-case performance. Obtaining such bounds is an interesting open problem. Also, it may be reasonable to assume that additional information on the clip retrieval periods is known. For example, the scheduler may have access to the (canonical) *prime factorizations* [16] of all periods. In other situations, the scheduler may be allowed some flexibility in the sense that it can *modify* the given clip retrieval periods within prespecified bounds (e.g., $\pm$ 20%). A natural question that arises is whether it is possible to design better heuristic algorithms that take advantage of such extra information and scheduling flexibilities.

(4) *Incorporating Different Retrieval Strategies.* The simple model of round-based C-SCAN retrieval with a fixed common round length T described in Section 2 is just one possible retrieval strategy. Many other strategies that have been proposed in the literature (e.g., the GSS scheme [7] that tries to optimize buffer usage) can be readily incorporated into our framework. More complicated problems arise when the differences in the clip display rates are such that *different round lengths* have to be used for different clips in order to use disks and memory effectively. For example, clips with small rates should use large round lengths to reduce the disk bandwidth wasted to latency overheads, whereas clips with large rates should use smaller round lengths to reduce their buffer requirements. Handling different round lengths requires significant and non-trivial extensions to our scheduling tree framework and methodology.

(5) *Dealing with Variable Bit-Rate (VBR) Data.* The schemes presented in tis paper can deal with VBR data using standard techniques. For example, the server can push data for clip C_i at a constant rate r_i and sufficient buffer space is allocated at the client to smooth variations with respect to r_i.

[9] Of course, the same approach can be used for other potential resource bottlenecks, e.g., network bandwidth.

Corollary 3.1 Determining whether there exists a scheduling forest for P is equivalent to 1-PMSP, and, thus, it is $\mathcal{NP}$-complete in the strong sense. □

Given the above intractability result, we have proposed a novel heuristic algorithm for constructing scheduling trees for a given (multi)set of periods [10]. Our algorithm is based on identifying and incrementally maintaining *candidate nodes* for scheduling incoming periods.

Definition 3.3 An *internal* node n at level l is *candidate for period* n_i if and only if $\pi_{l-1}(n)|n_i$ and $\gcd\left(w(n), \frac{n_i}{\pi_{l-1}(n)}\right) \geq \frac{w(n)}{w(n)-e(n)}$.

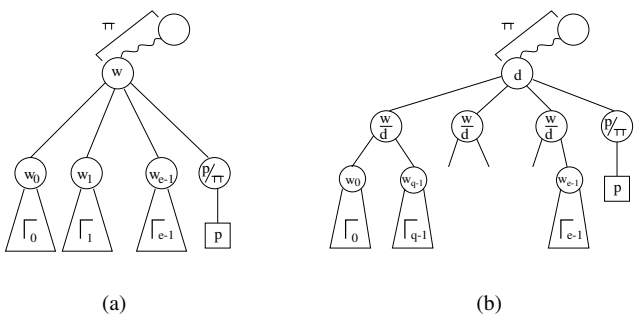

Figure 4: (a) Placing a period p under a tree node without splitting. (b) Period placement when the node is split.

A period n_i can be scheduled under any candidate node n in a scheduling tree. There are two possible cases: (1) *If* $\pi_l(n)|n_i$, then the condition in Definition 3.3 guarantees that n has at least one free edge at which n_i can be placed (Figure 4(a)). (2) *If* $\pi_l(n) \not| n_i$, then, in order to accommodate n_i under node n, n must be *split* so that the defining properties of the scheduling tree structure are kept intact. This is done as follows. Let $d = \gcd(w(n), \frac{n_i}{\pi_{l-1}(n)})$. Node n is split into a parent node with weight d and child nodes with weight $\frac{w(n)}{d}$, with the original children of n divided among the new child nodes, as shown in Figure 4(b); that is, the first batch of $\frac{w(n)}{d}$ children of n are placed under the first child node, and so on. It is easy to see that this splitting maintains the properties of the structure. Furthermore, the condition in Definition 3.3 guarantees that the new parent node will have at least one free edge for scheduling n_i.

The set of candidate nodes for each period to be scheduled can be maintained efficiently, in an incremental manner. The observation here is that when a new period n_i is scheduled, all remaining periods only have to check a maximum of three nodes, namely the two closest ancestors of the leaf for n_i and, if a split occurred, the last child node created in the split, for possible inclusion or exclusion from their candidate sets. Our heuristic algorithm (termed BUILDTREE) uses these observations to build a scheduling tree incrementally in a "greedy" fashion. The time complexity of BUILDTREE is $O(N^3)$, where N is the number of periods. For a detailed discussion, the interested reader is referred to [10].

3.3.2 Extensions for EPPV Scheduling

The periodic task scheduling model corresponding to EPPV service under Coarse-grained Striping is strictly more general than the model of simple periodic tasks considered in the previous section. As a consequence, two important extensions are necessary to the already described scheduling tree framework:

(1) *Scheduling Equidistant Subtasks.* The subtasks of C_i are periodic with period n_i, but they can not be scheduled independently. The scheduling algorithm needs to ensure that *all* the subtasks are scheduled together, using time slots (i.e., rounds) placed regularly at intervals of n_{disk}.

(2) *Handling Slots with Multi-Task Capacities.* The time slots at the leaves of a scheduling tree are *retrieval rounds* that, in general, can accommodate more than one subtask. Thus, the scheduling algorithm can allow for collisions to occur, as long as the total requirement (i.e., "size") does not exceed the length of a round.

Both these issues are addressed in [10], where we also describe a "packing-based" scheme for combining independent scheduling trees to improve slot utilization. An important observation is that the periodic scheduling framework developed for Coarse-grained Striping can also provide significant improvements for Clustering and Fine-grained Striping, when the retrieval periods of clips are larger than their length. More details can be found in [10].

3.4 Summary of Experimental Results

We have conducted an experimental study with the goal of comparing the average case performance of the proposed schemes for supporting EPPV service [10]. Our study was based on randomly generated workloads modeling typical situations encountered in today's pay-per-view video servers. The results of our experiments can be summarized as follows: (1) The performance Fine-grained Striping suffers from excessive disk latencies that severely limit the scheme's scalability as the size of the disk array grows. (2) The performance of Clustering, although superior to that of Fine-grained Striping, can still suffer from the effects of resource fragmentation leading to severe underutilization of the server for certain workload combinations. (3) The

collide during a round, their total bandwidth requirements do not exceed the capacity of the disk.

3.3.1 Periodic Maintenance and the Scheduling Tree Structure

The *k-server Periodic Maintenance Scheduling Problem (k-PMSP)* [5] is a special case of the problem of scheduling simple periodic tasks in a hard real-time environment. Briefly, the k-PMSP decision problem can be stated as follows: *Let $C = \{C_1, \ldots, C_N\}$ be a set of periodic tasks with corresponding periods $P = \{n_1, \ldots, n_N\}$, where each n_i is a positive integer. Is there a mapping of the the tasks in C to positive integer time slots such that successive occurrences of C_i are* **exactly** *n_i time slots apart and no more than k tasks ever collide in a slot?* Note that if u_i is the index of the first occurrence of C_i in a schedule for P then the (multi)set of starting time slots $\{u_1, \ldots, u_N\}$ uniquely determines the schedule, since C_i occurs at all slots $u_i + j \cdot n_i$, $j \geq 0$.

Baruah et al. [5] have shown that for any fixed value $k \geq 1$, k-PMSP is $\mathcal{NP}$-complete in the strong sense. Consequently, given a collection of simple periodic tasks with periods P, determining the existence of a collision-free schedule is intractable (i.e., it is equivalent to 1-PMSP). The existence of a *scheduling tree* structure (as described below) that contains all the periods in P, guarantees the existence of a collision-free schedule. Furthermore, the starting time slot for each task can be determined from the scheduling tree[7].

Definition 3.1 A *scheduling tree* is a tree structure consisting of nodes and edges with integer weights, where:

(1) Each internal node with weight w can have *at most* w outgoing edges, each of which has a distinct weight in $\{0, 1, \ldots, w-1\}$; and,

(2) Each leaf node represents a period n_i such that n_i is equal to the product of weights of the leaf's ancestor nodes.

We define the *level of a node (*or, *edge)* as the number of its proper ancestor nodes. Thus, the level of the tree's root is 0 and the level of all edges emanating from the root is 1. For any node n, let $w(n)$ and $e(n)$ denote the weight and the number of edges of n, respectively. Also, let $\texttt{anc_node}_j(n)$ represent the weight of the ancestor node of n at level j, and let $\texttt{anc_edge}_j(n)$ denote the weight of the ancestor edge of n at level j, where $j \leq level(n)$. Finally, define $\pi_j(n) = \prod_{i=0}^{j} \texttt{anc_node}_i(n)$ for $0 \leq j \leq level(n)$.

[7] To the best of our knowledge, no similar notion of tree structure for periodic task scheduling has been proposed in the real-time scheduling literature [37].

Consider a leaf node for period n_i located at level l. The first slot u_i in which the corresponding task is scheduled is defined from the scheduling tree structure as follows:

$$u_i = \texttt{anc_edge}_1(n_i) + \sum_{j=2}^{l} \texttt{anc_edge}_j(n_i) \cdot \pi_{j-2}(n_i). \quad (6)$$

Some intuition for the scheduling tree structure and the above formula is provided in Figure 3. The basic idea is that all tasks in a subtree rooted at some edge emanating from node n at level l will utilize time slot numbers that are congruent to $i \pmod{\pi_l(n)}$, where i is a unique number between 0 and $\pi_l(n) - 1$. Satisfying this invariant recursively at every internal node ensures the avoidance of collisions.

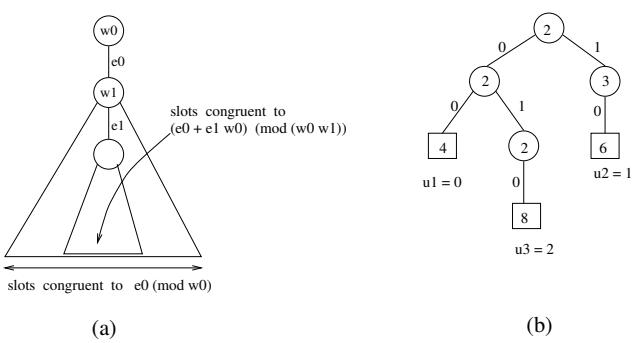

Figure 3: (a) The scheduling tree structure. (b) A tree for the set of periods $\{4, 6, 8\}$.

Note that the existence of a scheduling tree for a set of periods P is only a *sufficient condition* for the existence of a collision-free schedule. For example, the periods 6, 10, and 15 are schedulable using start times of 0, 1, and 2, respectively, although no scheduling tree can be built (since $\text{gcd}(\{6, 10, 15\}) = 1$)[8]. However, using the *Generalized Chinese Remainder Theorem* [16], it is straightforward to show that the existence of a *scheduling forest*, as defined below, is both necessary and sufficient for the existence a collision-free schedule.

Definition 3.2 Let $\Gamma_1, \ldots, \Gamma_k$ be scheduling trees for $P_1, \ldots, P_k$, where $P_1, \ldots, P_k$ is a partitioning of of the periods in P. The trees Γ_i and Γ_j are *consistent* if and only if for each $n_m \in P_i$ and $n_l \in P_j$ we have $u_m \not\equiv u_l \pmod{\text{gcd}(n_m, n_l)}$. A *scheduling forest* for P is a collection of pairwise consistent scheduling trees for some partition of P.

[8] The gcd() function returns the *greatest common divisor* of a set of integers.

0/1 knapsack optimization problem [14, 17, 33]. Thus, it is clearly $\mathcal{NP}$-hard [6]. Given the intractability of the problem, we have proposed (in [10]) a fast heuristic algorithm (termed PACKCLIPS) that combines the value density heuristic rule for the classical knapsack problem [9] with a First-Fit packing rule. Briefly, the main idea is to define the value density of clip C_i as the ratio $p_i = \frac{\texttt{value}(C_i)}{l(\mathbf{s}_i)}$, where $l(\mathbf{s}_i) = \max\{\texttt{size}_1(C_i), \texttt{size}_2(C_i)\}$, and pack the clips in decreasing order of density into unit capacity bins using a First-Fit rule. The schedulable subset (and the corresponding schedule) is determined by selecting the n_{disk} "most valuable" bins from the final packing. Algorithm PACKCLIPS can be implemented to run in time $O(N(\log N + n_{disk}))$. A detailed description can be found in [10]. The following theorem provides an upper bound on the worst-case performance ratio of our heuristic under a natural restriction on the storage requirements of a clip.

Theorem 3.1 ([10]) Assuming that the storage requirements of any clip are always less than or equal to one half of a disk's storage capacity, the 2-dimensional PACKCLIPS heuristic is 1/4-approximate; that is, if V_{OPT} is the value of the optimal schedulable subset and V_H is the value of the subset returned by PACKCLIPS then $\frac{V_H}{V_{OPT}} \geq \frac{1}{4}$. □

We have already noted that Clustering can lead to disk storage and bandwidth fragmentation, and this is clearly demonstrated in the rather discouraging worst-case bound of Theorem 3.1 – for "bad" lists of clips, PACKCLIPS may be able to utilize only as little as one fourth of the raw server capacity. Since storage fragmentation is not an issue when striping is used, we can effectively ignore storage constraints by assuming that the aggregate storage requirements of the clips to be scheduled do not exceed the storage capacity of the server; that is, we assume that $\sum_i l_i \cdot r_i \leq n_{disk} \cdot c_{disk}$ in the remainder of this section.

3.2 Fine-grained Striping

Our approach is based on the observation that, by virtue of the Fine-grained data organization, each clip imposes a constant load per round on *all* disks in the server (Figure 2(a)). Consequently, the problem of maximizing the effectively scheduled bandwidth clips under Fine-grained Striping corresponds to a traditional, single-bin, 0/1 knapsack problem with (one-dimensional) clip sizes $\texttt{size}(C_i) = \frac{\frac{d_i}{r_{disk} \cdot n_{disk}} + t_{lat}}{T - 2 \cdot t_{seek}}$ and values $\texttt{value}(C_i) = \left\lceil \frac{l_i}{T_i} \right\rceil \cdot r_i$. This is clearly a much simpler version of the knapsack model developed for Clustering and traditional knapsack heuristics can be used to provide near-optimal solutions [14, 17, 33]. In fact, a *one-dimensional* variant of PACKCLIPS (with number of bins/disks equal to 1) readily provides a 1/2-approximate heuristic for the problem [10].

[6] Note that the traditional knapsack problem remains $\mathcal{NP}$-hard even if the size of each item is equal to its value (i.e., the SUBSET SUM problem [9]). Thus, the fact that $\texttt{size}_1(C_i)$ and $\texttt{value}(C_i)$ are correlated does not affect the hardness of the problem.

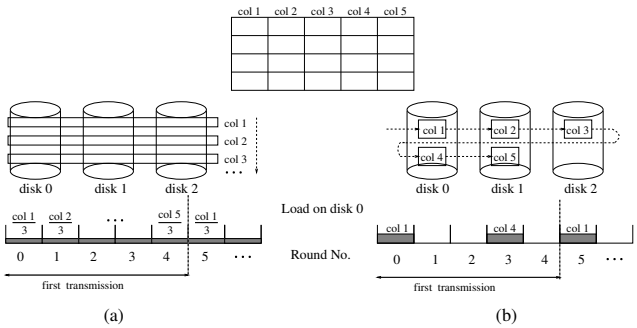

Figure 2: (a) Fine-grained Striping. (b) Coarse-grained Striping.

3.3 Coarse-grained Striping

Consider the retrieval of a clip matrix C_i from a particular disk in the array under Coarse-grained Striping (Figure 2(b)). By virtue of the round-robin placement, during each transmission of C_i, a column of C_i must be retrieved from that disk periodically, at intervals of n_{disk} rounds. Each such retrieval requires a fraction $\frac{\frac{d_i}{r_{disk}} + t_{lat}}{T - 2 \cdot t_{seek}}$ of the disk's bandwidth. Furthermore, to support EPPV service, the transmissions of C_i are themselves periodic with a period $T_i = n_i \cdot T$.

Thus, the retrieval of a clip matrix C_i from a specific disk in the array can be seen as a collection of *periodic real-time tasks* [20] with period T_i (i.e., the clip's transmissions), where each task consists of a collection of *subtasks* that are $n_{disk} \cdot T$ time units apart (i.e., column retrievals within a transmission). Moreover, the computation time of each such subtask is $\frac{d_i}{r_{disk}} + t_{lat}$. Note that the maximum number of subtasks mapped to a disk by C_i equals $\left\lceil \frac{c_i}{n_{disk}} \right\rceil$. ($c_i$ is the number of columns in C_i.) This number may actually be smaller for some disks in the array. However, in order to provide deterministic service guarantees for all disks, we consider only this worst-case number of subtasks in our scheduling formulation.

We say that two (or more) clip retrievals *collide* during a round if they are all reading data off the same disk. Collisions play a crucial role in our scheduling problem. Our algorithms need to ensure that whenever multiple retrievals

that cannot be reduced to simple algebraic conditions like Inequalities (2) and (4). A concise description of these problems and our proposed solution is given in Section 3.

2.3 Reducing Disk Latencies: Matrix-Based Allocation

By definition, the EPPV service model associates with each clip C_i a retrieval period T_i. We assume that retrieval periods are multiples of the round length T. This is a reasonable assumption, since retrieval periods will typically be multiples of minutes or even hours and the length of a round (usually bounded by buffering constraints) will not be more than a few seconds. The *matrix-based allocation scheme* [22, 27], increases the number of clients that can be serviced under the EPPV model by laying out data based on the knowledge of retrieval periods. The basic idea is to distribute, for each clip C_i, the starting points for the $\left\lceil \frac{l_i}{T_i} \right\rceil$ concurrent *display phases* of C_i uniformly across its length. Each such display phase corresponds to a different stream servicing (possibly) multiple clients. Conceptually, each clip C_i is viewed as a matrix consisting of elements of length T (Figure 1(a)).

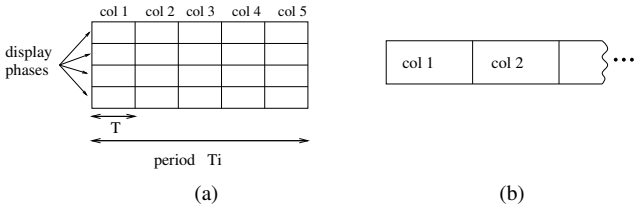

Figure 1: (a) A clip matrix. (b) Its layout on disk.

We define $n_i = \frac{T_i}{T}$ (i.e., the length of the retrieval period of C_i in rounds). The matrix for C_i consists of $c_i = \min\{n_i, \lceil \frac{l_i}{T} \rceil\}$ columns and $\left\lceil \frac{l_i}{T_i} \right\rceil$ rows (corresponding to the clip's display phases). Note that we can have $c_i < n_i$ when the retrieval period of the clip exceeds its length (i.e., $l_i < T_i$). Finally, we let d_i denote the amount of data in a column of C_i's matrix, that is $d_i = \left\lceil \frac{l_i}{T_i} \right\rceil \cdot T \cdot r_i$.

To support periodic retrieval, a clip matrix is stored in column-major form (i.e., data in each column is stored contiguously on disk) and its retrieval is performed *in columns* (i.e., one column per round) with each element handed to a different display phase (Figure 1(b)). Matrix-based allocation reduces the overhead of disk latency per stream since, in each round, it incurs a total overhead of only t_{lat} for $\left\lceil \frac{l_i}{T_i} \right\rceil$ streams of C_i, rather than $\left\lceil \frac{l_i}{T_i} \right\rceil \cdot t_{lat}$ (using Inequalities (1), (2), and (4)). This means that a single disk using the matrix-based scheme can support the periodic retrieval of $C_1, \ldots, C_k$ provided that the following inequality holds:

$$\sum_{\{C_i\}} \left(\frac{d_i}{r_{disk}} + t_{lat} \right) + 2 \cdot t_{seek} \leq T. \qquad (5)$$

The disk bandwidth *effectively utilized* by a clip during a round is the amount of raw disk bandwidth consumed by the clip without accounting for the latency overhead. For C_i, this is exactly $\frac{d_i}{T}$, or, equivalently, $\left\lceil \frac{l_i}{T_i} \right\rceil \cdot r_i$.

The matrix-based allocation scheme benefits all three multi-disk data organization strategies considered in this paper by significantly reducing the latency overhead in each round. For example, in Inequalities (2) and (4) for Clustering and Fine-grained Striping the term t_{lat} is replaced by $t_{lat} \cdot \left\lceil \frac{l_i}{T_i} \right\rceil^{-1}$. Thus, we present our results assuming that matrix-based allocation is used. For Fine-grained (Coarse-grained) Striping, this means that the retrieval unit striped (resp. distributed) across the disks of the server is an entire *column* of C_i rather than a single matrix element. However, we should stress that the schemes presented in this paper are equally applicable to the original data organization methods without the matrix-based scheme optimization.

3 EPPV Resource Scheduling: Algorithms and Performance

3.1 Clustering

Under a Clustered data organization, the EPPV resource scheduling problem reduces to effectively mapping clip matrices onto the server's disks so that the bandwidth and storage requirements of each matrix are satisfied. We associate two key parameters with each clip:

(1) A 2-dimensional *size vector* $\mathbf{s}_i = [\mathtt{size}_1(C_i), \mathtt{size}_2(C_i)]$ where the first component is the normalized bandwidth consumption of the clip matrix and the second component is the normalized storage capacity requirement of the clip. More formally,

$$\mathtt{size}_1(C_i) = \frac{\frac{d_i}{r_{disk}} + t_{lat}}{T - 2 \cdot t_{seek}} \quad \text{and} \quad \mathtt{size}_2(C_i) = \frac{l_i \cdot r_i}{c_{disk}}.$$

(2) A *value*: $\mathtt{value}(C_i) = \left\lceil \frac{l_i}{T_i} \right\rceil \cdot r_i$, that corresponds to the bandwidth *effectively utilized* by C_i during a round.

Using these definitions, the problem of maximizing the effectively scheduled disk bandwidth can be formally stated as follows: *Given a collection of clips $C = \{C_1, \ldots, C_N\}$, determine a subset C' of C and a packing of the vectors $\{\mathbf{s}_i : C_i \in C'\}$ in n_{disk} unit capacity (i.e., $[1, 1]$) bins such that the total value $\sum_{C_i \in C'} \mathtt{value}(C_i)$ is maximized*. This problem is a generalization of the traditional

arbitrary locations are eliminated and the total seek overhead during a round is bounded by $2 \cdot t_{seek}$. Furthermore, retrieval of each non-contiguously stored piece of data can incur a disk latency overhead of at most t_{lat} during a round. To ensure continuous delivery for all streams during a round, the sum of the total disk transfer time for all data retrieved and the overall latency and seek time overhead cannot exceed the length T of the round [11, 32]. More formally, we require the following inequality to hold[4]:

$$\sum_{\{stream(C_i)\}} \left(\frac{T \cdot r_i}{r_{disk}} + t_{lat}\right) \leq T - 2 \cdot t_{seek}. \quad (1)$$

2.2 Multi-Disk Data Organization Schemes

2.2.1 Clustering

In Clustering, each disk of a multi-disk system is viewed as an autonomous unit – entire clips are stored on and retrieved from a single disk and multiple clips can be clustered on each disk. For our round-based model of data retrieval, this means that Inequality (1) must be satisfied on each disk, where the summation is taken over all streams retrieving clips C_i stored on that disk. Note that by the definition of EPPV service, the number of concurrent streams retrieving clip C_i is exactly $\left\lceil \frac{l_i}{T_i} \right\rceil$. Furthermore, each magnetic disk has a limited storage capacity that cannot be exceeded by the set of stored clips. Thus, if we let $\{C_i\}$ denote the set of clips clustered on any single disk in the server, we require the following conditions to be satisfied:

$$\left\lceil \frac{l_i}{T_i} \right\rceil \cdot \left[\sum_{\{C_i\}} \left(\frac{T \cdot r_i}{r_{disk}} + t_{lat}\right) \right] \leq T - 2 \cdot t_{seek} \quad (2)$$

$$\text{and} \quad \sum_{\{C_i\}} l_i \cdot r_i \leq c_{disk}. \quad (3)$$

2.2.2 Fine-grained Striping

A major deficiency of clustered data organization for large scale continuous media services is that it can lead to bandwidth *and* storage fragmentation, and, consequently, underutilization of server resources. Striping schemes eliminate storage fragmentation by declustering a clip's data across all available disks. This essentially means that we no longer need to satisfy a storage capacity constraint on each disk, as long as the total storage requirements of the clips to be scheduled do not exceed the storage capacity of the disk array.

In Fine-grained Striping [25], each retrieval unit of a clip is striped across all n_{disk} disks of the server. Consequently, every stream retrieval involves all the n_{disk} disk heads working in parallel, with each disk being responsible for fetching $\frac{T \cdot r_i}{n_{disk}}$ bits of C_i in each round. This striping strategy is employed in the RAID-3 data distribution scheme [30]. For EPPV service, the number of concurrent streams retrieving clip C_i from the array is exactly $\left\lceil \frac{l_i}{T_i} \right\rceil$. Thus, to ensure continuous delivery, the following condition must be satisfied:

$$\left\lceil \frac{l_i}{T_i} \right\rceil \cdot \left[\sum_{i=1}^{N} \frac{T \cdot r_i}{r_{disk} \cdot n_{disk}} + N \cdot t_{lat} \right] \leq T - 2 \cdot t_{seek}, \quad (4)$$

where N is the total number of clips on the server (see Table 1). Despite its conceptual simplicity, Fine-grained Striping can lead to underutilization of available disk bandwidth due to increased latency overheads [25]. This is clearly demonstrated in the above condition for continuity, which shows that, during each round, all disks incur a penalty of t_{lat} for each clip stored *in the entire server*. These latency penalties limit the scalability of an EPPV server based on Fine-grained Striping since the problem is obviously exacerbated as the size of the disk array grows.

2.2.3 Coarse-grained Striping

In the Coarse-grained Striping scheme [25], the retrieval units of each clip are mapped to individual disks in a round-robin manner. Consequently, the retrieval of data for a stream on clip C_i proceeds in a round-robin fashion along the disk array. During each round, a single disk is used to fetch a retrieval unit of C_i and consecutive rounds employ consecutive disks[5]. This striping strategy is employed in the RAID-5 data distribution scheme [30].

Coarse-grained Striping avoids the large latency overheads of the Fine-grained scheme and, consequently, can offer much better scalability and bandwidth utilization [25]. On the other hand, supporting periodic stream retrieval requires much more sophisticated scheduling methods than either Clustering or Fine-grained Striping. This is because, unlike the two previous data organization schemes, Coarse-grained Striping does not impose a steady load on each disk *during each round*. Consider the retrieval of clip C_i from a particular disk in the array. By virtue of the round-robin placement, each stream retrieving data of C_i needs to fetch a retrieval unit of $T \cdot r_i$ bits from that disk periodically, at intervals of n_{disk} rounds. Furthermore, to support EPPV service, the streams retrieving clip C_i are themselves periodic with a period $T_i = n_i \cdot T$. Thus, supporting continuous, periodic service under Coarse-grained Striping gives rise to complex *periodic real-time task scheduling problems* [20]

[4] Although our schemes can be extended to handle disk calibration and multi-zone disks [24], these issues are not addressed in this paper.

[5] We assume that a disk has sufficient bandwidth to support the retrieval of one or more clips. If this does not hold, one or more disks can be viewed as a single composite disk.

In each case, our main objective has been to maximize the amount of disk bandwidth that is effectively scheduled under the given organization and storage constraints. This is typically the situation facing large-scale CM servers that periodically need to re-schedule their offerings to adapt to a changing audience, content, and popularity profile[3][19, 31]. Briefly, our main results can be stated as follows:

(1) For Clustering and Fine-grained Striping, we have formulated EPPV resource scheduling problems as generalized variants of the 0/1 knapsack problem [14, 17, 33]. Since the problems are clearly $\mathcal{NP}$-hard, we have proposed provably near-optimal heuristics with low polynomial-time complexity.

(2) For Coarse-grained Striping, we have shown that the resource scheduling problems involved in supporting the EPPV service model are non-trivial generalizations of the Periodic Maintenance Scheduling Problem (PMSP) [38]. Given that PMSP is known to be $\mathcal{NP}$-complete in the strong sense [5], we have proposed heuristic scheduling algorithms based on the novel concept of a *scheduling tree* for Periodic Maintenance. Thus, our work has also contributed to the area of hard real-time scheduling theory by proposing the scheduling tree structure and algorithms as a new approach to PMSP.

The results of our experimental study have confirmed the superiority of the "Coarse-grained Striping + Scheduling Trees" combination under different workloads [10]. We believe that our proposed scheduling tree framework is a powerful tool, applicable to any scenario involving regular, periodic use of a shared resource. A prototype implementation of the framework is currently being built on top of the *Fellini* multimedia storage server at Bell Labs [21].

2 Notation and System Model

In this section we present a brief overview of our CM server model and the notation that will be used throughout this paper. First, we present a model of *round-based retrieval* for continuous media. Our model follows the conventions used in most earlier work on multimedia storage servers [7, 11, 24, 32]. Next, we present the three different multi-disk data organization schemes considered in this paper (Clustering, Fine-grained and Coarse-grained Striping) and describe how the round-based retrieval model adapts to each different organization scenario. Finally, we describe the *matrix-based allocation scheme* [26, 27] that tries to optimize the data layout of a clip based on the knowledge of its retrieval period. Table 1 summarizes the notation used in this paper with a brief description of its semantics. Additional notation will be introduced when necessary.

Parameter	Semantics
C_i	Continuous media clip ($i = 1, \ldots, N$) (also, task of retrieving C_i)
r_i	Display rate for clip C_i (in Mbps)
T_i	Retrieval period for clip C_i (in sec)
T	Time unit of clip retrieval (round length)
l_i	Length of clip C_i (in sec)
n_i	Retrieval period of C_i in rounds
c_i	Number of columns in the matrix of C_i
d_i	Maximum column size (in bits)
n_{disk}	Number of disks in CM server
r_{disk}	Disk transfer rate
c_{disk}	Disk storage capacity
t_{seek}	Disk seek time
t_{lat}	Disk latency

Table 1: Clip and Disk Parameters

2.1 Retrieving Continuous Media Data

Consider a single magnetic disk storing a collection of continuous media clips. We assume that the disk has a transfer rate of r_{disk}, a storage capacity of c_{disk}, a worst-case seek time of t_{seek}, and a worst-case latency of t_{lat} (which consists of rotational delay and settle time). A clip C_i is characterized by a display rate r_i (the rate at which data for C_i must be transmitted to clients) and a length l_i (in units of time). We refer to the transmission of a clip starting at a given time as a *stream*. Data for streams is retrieved from the disk in *rounds of length* T. For a stream displaying clip C_i (denoted by $stream(C_i)$), a circular buffer of size $2 \cdot T \cdot r_i$ is reserved in the server's buffer cache. In each round, while the stream is consuming $T \cdot r_i$ bits of data from its buffer, the $T \cdot r_i$ bits that the stream will consume in the next round are retrieved from the disk into the buffer. This ensures that each stream will have sufficient data to display the corresponding clip continuously. The quantity $T \cdot r_i$ is termed the *retrieval unit* of clip C_i.

During a round, for streams $stream(C_1), \ldots, stream(C_k)$ for which data is to be retrieved from disk, $T \cdot r_1, ..., T \cdot r_k$ bits are read using the C-SCAN disk head scheduling algorithm [36]. C-SCAN scheduling ensures that the disk heads move in a single direction when servicing streams during a round. As a result, random seeks to

[3] A related problem is *capacity planning:* Given a fixed user profile, determine the minimum system configuration that can accommodate it. This problem is essentially dual to ours, and we believe that most of our techniques are applicable. We will not discuss capacity planning further in this paper.

Work on batching has typically concentrated on different disciplines for scheduling requests from the server's queue. The goal is to strike a balance between: (1) fairness, (2) minimizing average waiting time, and (3) minimizing client reneging probability (i.e., the probability that a client's request is cancelled due to excessive waiting). Scheduling disciplines like FCFS, Maximum Queue Length (MQL), and Maximum Factored Queue Length (MFQ) have been proposed and evaluated using simulation models [2, 8]. The main problem with these batching policies is that they are *unpredictable*, in the sense that they cannot offer a guaranteed upper bound on how long a client request must wait in the queue. Furthermore, results on the behavior of such policies (either analytical or by simulation) are typically based on specific probabilistic models of "customer reneging behavior" whose accuracy is often questionable in practice. Buffering is based on the idea that it is possible to trade extra memory for reduced bandwidth demand. This is a very general approach that is orthogonal to other data sharing schemes and, consequently, can be incorporated into batching, piggybacking, or EPPV, as an additional optimization (e.g., to facilitate VCR functionality). However, we should note that with current hardware pricing and stream parameters, trading memory for disk bandwidth is often a losing proposition [18]. Piggybacking uses the fact that small differences in the display rates (e.g., deviations of at most 5% from the normal display rate) are not perceived by the viewer. Nevertheless, the idea raises difficult implementation issues when MPEG-like compression is used, because of inter-frame dependencies.

Compared to other data sharing schemes (most notably batching), EPPV service offers the advantage of *predictability* – the response time for transmission of a clip to a client is bounded by the clip's retrieval period[2]. This retrieval period is typically determined by the service provider, based on factors such as clip popularity, legal/financial constraints on the distribution rights of a clip, and specific programming choices. Because of the regular pattern of clip retrievals, clients can be informed of the exact time that a specific transmission will start. Thus, even when resources are scarce the EPPV service model can guarantee *predictable* response times for all incoming requests. An additional benefit of the regularity of EPPV service is that it can also support user interaction through VCR-like operations [4, 26]. From the service provider's perspective, a desirable feature the regular structure of EPPV service is that it simplifies the periodic scheduling of live events, such as news and sports events, into the program.

Because of the advantages outlined above and its potential to provide scalable, cost-effective CM offerings, EPPV is quickly becoming the service model of choice for telecom, cable, broadcast, and content companies [31]. Realizing this potential, however, requires schemes for effectively scheduling the available disk bandwidth and storage capacity so that high levels of concurrency and system utilization can be sustained. Two phenomena make this a challenging problem — the periodic nature of EPPV service and the relatively high latencies of magnetic disk storage. The periodicity of clip retrievals in EPPV servers generates a host of difficult periodic task scheduling problems that fall within the realm of hard real-time scheduling theory [20]. The high disk latencies complicate effective utilization of disk bandwidth and storage with reasonable amounts of buffer space, which is an important cost factor in CM server design. The use of multiple disks to handle the high storage volume and bandwidth requirements of CM data exacerbates the problem. Thus, the need for intelligent scheduling mechanisms becomes more pronounced as the scale of the system increases.

Despite the importance of the resource scheduling problem for EPPV service, prior work has typically concentrated on other issues such as data layout schemes to efficiently support periodic retrieval [22, 23], and support for VCR functionality under EPPV [4, 18]. Abram-Profeta and Shin [1] used simple queueing models to solve the problem of assigning optimal retrieval periods to the set of clips stored at an EPPV server. Their models assume that all clips have the *same length and display rate requirement* and ignore multi-disk data organization issues. None of the above efforts has considered the general problem of EPPV resource scheduling and, consequently, they can be viewed as orthogonal to our work. A restricted form of the problem that assumes all clips to have *identical* display rates and similar lengths was addressed in the work of Özden et al. [28].

In this paper, we outline our recent results on the *general* resource scheduling scheduling problem for EPPV service [10] and we present a number of interesting research directions that we are currently pursuing. The scheduling framework we have developed in [10] handles continuous media data with (possibly) different display rates (depending on the media type and/or the compression scheme), different periods (e.g., depending on the popularity of a clip), and arbitrary lengths. Given a hardware configuration and a collection of clips to be scheduled, we have presented schemes for determining a schedulable subset of clips under three different data organization schemes for multiple disks: *Clustering*, *Fine-grained Striping*, and *Coarse-grained Striping*.

[2] In existing television terminology, the term "Pay-Per-View" refers to both a prescheduled playback program and a certain pricing mechanism. We refer to our service model as EPPV to emphasize the scheduling aspect of the service without constraining its pricing mechanism.

In *Proceedings of the 1998 IEEE RIDE Workshop (Continuous-Media Databases and Applications),* Orlando, Florida, February 1998, pp. 111–120.

On Periodic Resource Scheduling for Continuous Media Databases

Minos N. Garofalakis*
Bell Laboratories
minos@research.bell-labs.com

Banu Özden
Bell Laboratories
ozden@research.bell-labs.com

Avi Silberschatz
Bell Laboratories
avi@research.bell-labs.com

Abstract

The Enhanced Pay-Per-View (EPPV) service model provides an effective way of delivering continuous media (CM) data clips to a large number of different clients. The aim is to share CM streams among various clients, thereby increasing the level of concurrency beyond the capacity limitations of available resources (e.g., disk bandwidth, RAM, and network bandwidth), while guaranteeing an upper bound on the clients' waiting time. In the EPPV service model, waiting time guarantees are provided by means of a regular playback schedule, where each continuous media clip is displayed periodically *with a frequency that depends on factors such as the clip's popularity. Efficiency is achieved by sharing these periodic CM streams among multiple clients. The EPPV model offers a number of advantages over other schemes (e.g., batching), which makes it more attractive to large-scale service providers. The purpose of this paper is twofold: (1) to provide an overview of our recent results on the resource scheduling problems associated with supporting EPPV for continuous media clips with (possibly) different display rates, frequencies, and lengths; and, (2) to outline a number of open research problems related to the EPPV service model. A major contribution of our work is the introduction of a robust scheduling framework that, we believe, can provide solutions for a variety of realistic EPPV resource scheduling scenarios, as well as any scheduling problem involving regular, periodic use of a shared resource. Based on this framework, we propose various interesting research directions for extending the results presented in this paper.*

1 Introduction

Next generation database systems will need to provide support for various forms of multimedia data such as images, video, and audio. Multimedia data types differ from conventional alphanumeric data in their characteristics, and hence require different techniques for their organization and management. Of these new data types, *Continuous Media* (CM) data (video and audio) present a major challenge since they consist of *streams* of media quanta (video frames or audio samples) that must be delivered in a timely fashion. Hence, in contrast to traditional storage managers, a CM storage server needs to ensure that the retrieval and storage of such CM streams proceed at their pre-specified real-time rates.

A CM server, like any other storage server, has a limited amount of resources (e.g., memory, disk/network bandwidth, and disk storage), which places a hard limit on the number of streams that can be simultaneously delivered. Thus, it is a challenging problem to design effective resource management algorithms that can provide on-demand support for a large number of concurrent continuous media clients. For many application environments, like Movies-on-Demand, a CM server typically needs to sustain levels of concurrency that far exceed the limits imposed by available resource capacities. This means that the conventional *Random Access* (or, *Fully Interactive*) service model that places resource reservations to allocate independent physical channels for each client, cannot possibly provide cost-effective solutions in such environments [22, 23]. As a result, several *data sharing* techniques have been proposed in the literature for increasing the number of concurrent clients beyond the capacity limitations of available resources:

- *Batching* [2, 3, 8, 34] allows several clients waiting in the server's queue for the same CM clip to share the same stream.
- *Buffering* (or, *Bridging*) [15, 29, 35] uses extra memory buffers in a controlled fashion to allow requests that arrive with a small phase difference with respect to the start of a stream to fetch their data blocks directly from memory (i.e., with no disk access).
- *Piggybacking* [12] allows clients to view the same clip at different display speeds so that, eventually, they can catch up with each other and share the same stream.
- *Enhanced Pay-Per-View (EPPV)* (or, *Periodic*) service [10, 22] assigns each clip a *retrieval period*, typically determined by the clip's popularity. Streams retrieving a clip are initiated periodically at offsets equal to the clip's retrieval period and multiple clients can share the same stream[1].

*A Ph.D. candidate at the University of Wisconsin–Madison.

[1]Depending on the underlying networking technology, clips can be delivered to clients via uni, multi, or broadcast channels.