Approximation Algorithms for Data Placement on Parallel Disks

L. Golubchik * S. Khanna † S. Khuller ‡ R. Thurimella § A. Zhu ¶

Abstract

We study an optimization problem that arises in the context of data placement in multimedia storage systems. We are given a collection of M multimedia data objects that need to be assigned to a storage system consisting of N disks $d_1, d_2 ..., d_N$. We are also given sets $U_1, U_2, ..., U_M$ such that U_i is the set of clients requesting the ith data object. Each disk d_j is characterized by two parameters, namely, its *storage capacity* C_j which indicates the maximum number of data objects that may be assigned to it, and a *load capacity* L_j which indicates the maximum number of clients that it can serve. The goal is to find a placement of data objects on disks and an assignment of clients to disks so as to maximize the total number of clients served, subject to the capacity constraints of the storage system.

We study this data placement problem for two natural classes of storage systems, namely, *homogeneous* and *uniform ratio*. Our first main result is a tight upper and lower bound on the number of items that can *always* be packed for any input instance to homogeneous as well as uniform ratio storage systems. We show that an algorithm given in [11] for data placement, achieves this bound. Our second main result is a polynomial time approximation scheme for the data placement problem in homogeneous and uniform ratio storage systems, answering an open question of [11]. Finally, we also study the problem from an empirical perspective.

1 Introduction

We study a *data placement problem* that arises in the context of multimedia storage systems. In this problem, we are given a collection of M multimedia data objects that need to be assigned to a storage system consisting of N disks $d_1, d_2 ..., d_N$. We are also given sets $U_1, U_2, ..., U_M$ such that U_i is the set of clients requesting the ith data object. Each disk d_j is characterized by two parameters, namely, its *storage capacity* C_j which indicates the maximum number of data objects that may be assigned to it, and its *load capacity* L_j which indicates the maximum number of clients that it can serve simultaneously. The goal is to find a placement of data objects on disks and an assignment of clients to disks so as to maximize the total number of clients served, subject to the capacity constraints. We can view each data object simply as a color, and each client as a unit size item with a color corresponding to the data object in which they are interested. Each disk d_j can hold at most L_j items, with the additional constraint that the total number of distinct colors of the items in d_j does not exceed C_j.

The data placement problem described above arises naturally in the context of storage systems for multimedia objects where one seeks to find a placement of the data objects, e.g., movies, on a system of disks. These and other issues related to the design of multimedia storage systems are discussed in [8]. We study this data placement problem for the following two natural types of storage systems.

Homogeneous Storage Systems: In a homogeneous storage system, all disks are identical. We denote by k and L the storage capacity and the load capacity, respectively, of each disk and refer to this variant as k-HDP (homogeneous data placement).

Uniform Ratio Storage Systems: In a uniform ratio storage system, the ratio L_j/C_j of the load to the storage capacity is identical for each disk. We denote by $C_{\min}$ and $C_{\max}$ the minimum and the maximum storage capacity of any disk in such a system and refer to this variant as URDP (uniform ratio data placement). Notice that homogeneous storage systems are a special case of uniform ratio storage systems.

In the remainder of this paper, we assume that (i) the total number of clients does not exceed the total load capacity, i.e., $\sum_{i=1}^{M} |U_i| \leq \sum_{j=1}^{N} L_j$, and (ii) the total number of data objects does not exceed the total storage capacity, i.e., $M \leq \sum_{j=1}^{N} C_j$.

1.1 Related Work

The data placement problem described above bears some resemblance to the classical multi-dimensional

*Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Research supported by NSF CAREER Award CCR-9896232. E-mail : leana@cs.umd.edu.

†Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104. This work was done when the author was at Bell Laboratories, Murray Hill, NJ 07974. Email : sanjeev@cis.upenn.edu.

‡Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Research supported by NSF Award CCR-9820965 and NSF CAREER Award CCR-9501355. E-mail : samir@cs.umd.edu.

§Department of Computer Science, The University of Denver, Denver, CO 80208. This work was done while the author was visiting the University of Maryland. E-mail : ramki@cs.du.edu.

¶Computer Science Department, Stanford University, Stanford, CA, 94305. This work was done while the author was with the Department of Computer Science at the University of Maryland. E-mail : anzhu@cs.stanford.edu.

knapsack problem [6, 10, 2]. We may view each disk as a knapsack with a load as well as a storage dimension, and each client as a unit size item with a color associated with it. However, in our problem, the storage dimension of a disk behaves in a *non-aggregating* manner in that assigning additional items of an already present color does not increase the load along the storage dimension. It is this particular aspect of our problem that makes it difficult to adapt known techniques for multi-dimensional packing problems.

Shachnai and Tamir [11] studied the above data placement problem; they refer to it as the *class constrained multiple knapsack* problem. The authors gave an elegant algorithm, called the *sliding window* algorithm, and showed that this algorithm packs all items whenever $\sum_{j=1}^{N} C_j \geq M + N - 1$ for URDP. An easy corollary of this result is that one can always pack a $(1 - \frac{1}{1+C_{\min}})$-fraction of all items for URDP. (This is seen by increasing the capacity of each disk by 1, packing all the items, and then from each disk dropping the color with the fewest items.) The authors showed that the problem is NP-hard when each disk has an arbitrary load capacity, and unit storage. Recently, Shachnai and Tamir [12] study a variation of the data placement problem above where in addition to having a color, each item u has a size $s(u)$ and a profit $p(u)$ associated with it. For the special case when $s(u) = p(u)$ for each item, and the *total* number of different colors M is *constant*, the authors give a dual approximation scheme whereby for any $\epsilon > 0$, they give a polynomial time algorithm to obtain a $(1-\epsilon/4)$-approximate solution provided the load capacity is allowed to be exceeded by a factor of $(1+\epsilon)$. The paper by Dawande, Kalagnanam and Sethuraman [5] deals with a related bin-packing problem, where there are n items, each with a color and size, and m distinct bin sizes. The objective is to pack all items into bins such that no bin contains items of more than p colors. The objective is to minimize the total capacity of the bins that are used. For the case where the total number of colors is constant they give a polynomial time approximation scheme (PTAS). The authors also describe some practical heuristics for the problem which give a constant approximation guarantee.

1.2 Our Results

Our first main result is a tight upper and lower bound on the number of items that can *always* be packed for any input instance to homogeneous as well as uniform ratio storage systems. It is worth noting that in the case of completely arbitrary storage systems no such absolute bounds are possible.

THEOREM 1.1. (Section 3) *It is always possible to pack a $(1 - \frac{1}{(1+\sqrt{k})^2})$-fraction of items for any instance of k-HDP, or more generally, a $(1-\frac{1}{(1+\sqrt{C_{\min}})^2})$-fraction of items can always be packed for any instance of URDP. Moreover, there exists a family of instances for which it is infeasible to pack any larger fraction of items.*

The upper bounds above are constructive as they are achieved by the sliding window algorithm of [11]. A side-result of our proof technique here is a simple alternate proof of the result that all items can be packed whenever $\sum_{j=1}^{N} C_j \geq M + N - 1$.

Our second main result is a polynomial time approximation scheme for the data placement problem in uniform ratio storage systems, answering an open question of [11].

THEOREM 1.2. (Section 4) *For any fixed $\epsilon > 0$, one can obtain in polynomial time a $(1-\epsilon)$-approximate solution to the data placement problem for any instance of URDP.*

We also strengthen the NP-hardness results of [11] by showing that the data placement problem is NP-hard even for very special cases of homogeneous storage systems.

THEOREM 1.3. (Appendix A) *The k-HDP problem is NP-complete for homogeneous disks with storage capacity $k = 2$ and strongly NP-hard for $k \geq 3$.*

Both reductions above are from NP-hard partitioning problems and illustrate how item colors can effectively encode large non-uniform sizes arising in the instances of these partitioning problems, even though each item in our problem is of unit size itself. We also note here that the case $k = 1$ is easily solvable in polynomial time.

Finally, we also study the problem from an empirical perspective. We study the homogeneous case on instances generated by a Zipf distribution [9] (this corresponds to *measurements* performed in [3] for a movies-on-demand application) and compare the actual performance of the sliding window algorithm with the bounds obtained above as well as the bounds in [11]. The results of this study are presented in Section 3.2. We also show how to implement the sliding window algorithm so that it runs in $O((N+M)\log(N+M))$ steps, improving on the $O(NM)$ running time of [11] (details of this implementation will be provided in the full version of the paper). Note that the input size for our problem is measured by N and M and not the total number of items (which can be much larger). The algorithm will output the subset of data objects assigned to each disk and the number of items of each color assigned to each disk.

We next describe in some detail the motivating application for our data placement problem.

1.3 Motivational Application

Recent advances in high speed networking and compression technologies have made multimedia services, such as video-on-demand (VOD) servers, feasible. The enormous storage and bandwidth requirements of multimedia data necessitates that such systems have very large disk farms. One viable architecture is a parallel (or distributed) system with multiple processing nodes in which each node has its own collection of disks and these nodes are interconnected, e.g., via a high-speed network.

We note that disks are a particularly interesting resource. Firstly, disks can be viewed as "multidimensional" resources — the dimensions being storage capacity and load capacity. Based on the application one or the other resource can be the bottleneck. Secondly, all disk resources are not equivalent since a disk's utility is determined by the data stored on it. It is this "partitioning" of resources (based on data placement) that contributes to some of the difficulties in designing cost-effective parallel multimedia systems, and I/O systems in general. In a large parallel VOD system improper data distribution can lead to a situation where requests for (popular) videos cannot be served even when the overall load capacity of the system is not exhausted because these videos reside on highly loaded nodes, i.e., the available load capacity and the necessary data are not on the same node.

One approach to addressing the load imbalance problem is to partition each video across all the nodes in the system and thus avoid the problem of "splitting resources", e.g., as in the staggered striping technique [1]. However, this approach suffers from a number of implementation-related shortcomings that are detailed in [4]. An alternate system is described in [14] where the nodes are connected in a shared-nothing manner [13]. Each node j has a finite storage capacity, C_j (*in units of continuous media (CM) objects*), as well as a finite load capacity, L_j (*in units of CM access streams*). These nodes are constructed by combining groups of disks into disk clusters. In fact, in this paper we will mostly view nodes as "logical disks". For instance, consider a server that supports delivery of MPEG-2 video streams where each stream has a bandwidth requirement of 4 Mbits/s and each corresponding video file is 100 mins long. If each node in such a server has 20 MBytes/s of load capacity and 36 GB of storage capacity, then each such node can support $L_j = 40$ simultaneous MPEG-2 video streams and store $C_j = 12$ MPEG-2 videos. In general, different nodes in the system may differ in their storage and/or load capacities.

In our system each CM object resides on one or more nodes of the system. The objects may be striped on an *intra-node* basis but *not* on the *inter-node* basis. For example, objects that require more than a single node's load capacity (to support the corresponding requests) are *replicated* on multiple nodes. The number of replicas needed to support requests for a CM object is a function of the demand. This should result in a scalable system which can grow on a node by node basis.

The difficulty here is in deciding on: (1) how many copies of each video to keep, which can be determined by the demand for that video, e.g., as in [14], and (2) how to place the videos on the nodes so as to satisfy the total anticipated demand for each video within the constraints of the given storage system architecture. Our data placement problem tries to capture these issues.

1.4 Organization

We start with an overview of the sliding window algorithm in Section 2. In Section 3, we present a tight analysis of the sliding window algorithm to derive the upper bounds of Theorem 1.1. We also present here a family of instances that give the matching lower bound as well as numerical results. Finally, in Section 4 we present our approximation schemes for homogeneous as well as uniform ratio storage systems and thus establish Theorem 1.2. A proof of Theorem 1.3 appears in Appendix A.

2 Sliding Window Algorithm

For sake of completeness, we describe here the sliding window algorithm of Shachnai and Tamir [11].

We maintain a list $R[1], \ldots, R[m]$, $1 \leq m \leq M$ of colors where $R[i]$ denotes the number of items of color i that remain. The list R is sorted in non-decreasing order. At step j, we assign items to disk d_j. For the sake of readability, $R[i]$ always refers to the number of *currently* unassigned items of a particular color (i.e., we do not explicitly indicate the current step j of the algorithm in this notation). Assume that the disks are numbered in a non-decreasing order of their capacities, i.e., $C_1 \leq C_2 \leq \cdots \leq C_N$. We assign items and remove from R the colors that are packed completely, and we move the partially packed colors to their updated places (in the sorted set) according to the remaining number of unpacked items of that color.

The assignment of colors to a disk d_j follows the general rule that we select the first consecutive sequence of C_j or less colors, $R[u], \ldots, R[v]$, whose total number of items either equals to or exceeds the load capacity L_j. We then assign colors $R[u], \ldots, R[v]$ to d_j. In order to not exceed the load capacity, we will split the items of the last color $R[v]$. It could happen that no such sequence of colors is available, i.e., all colors have relatively few items. In this case, we greedily select the colors with the largest number of items to fill the current disk. The selection procedure is as follows: we first examine $R[1]$, which is the color with the smallest number of items. If these items exceed the load capacity, we will assign $R[1]$ to the first disk and re-locate the remaining piece of $R[1]$ (which for R[1] will always be the beginning of the list). If not, we then examine the total number of items in $R[1]$ and $R[2]$, and so on until either we find a sequence of colors with a sufficiently large number of items ($\geq L_j$), or the first C_j colors have a total number of items $< L_j$. In the latter case, we go on to examine the next C_j colors $R[2], \ldots, R[C_j + 1]$ and so on, until either we find C_j colors with a total number of items at least L_j or we are at the end of the list, in which case we simply select the last sequence of C_j colors which has the greatest total number of items.

We note that the Sliding Window algorithm can be implemented to run in $O((N + M)\log(N + M))$ steps, where N is the number of disks and M is the number of colors. Note that this is a significant improvement on the running time in [11]. The reason for the improvement is that we do not have to start the "window", of length C_j, from the beginning of the list R in each iteration j. At the end of iteration i, after we have determined the colors $R[m], \ldots, R[n]$ to place in disk d_i, we know that $R[j] < \frac{L_i}{C_i} = r$ for all $j \leq m - 1$. This indicates that all "windows" ending with $R[j]$, for all $j \leq m - 1$ will under-fill the load of any disk. Thus the right end of the "window" will always monotonically move to the right. We need to use a 2-3 tree data structure to implement this algorithm. (Skip-lists can be used as well to get the same expected worst case running time, with a simpler implementation.) Details are defered to the full version.

3 Analysis

We now show that the Sliding Window Algorithm guarantees to pack $(1 - \frac{1}{(1+\sqrt{k})^2})$ fraction of all items in the homogeneous case. We assume each disk has load capacity L and storage capacity k. The Sliding Window Algorithm guarantees to pack $(1 - \frac{1}{(1+\sqrt{C_{\min}})^2})$ fraction of items in the uniform ratio case, where the minimum capacity of a disk is denoted by $C_{\min}$. We also show that these bounds are tight.

We first discuss the homogeneous case. Note that if there are any unpacked items, then every disk is filled to the maximum either on the number of items it can hold or on the number of colors that can be stored. We will call the former as *load saturated* and the latter (the rest) as *storage saturated*. (Therefore, if a disk is storage saturated, then it still has some unfilled load capacity.) Denote the number of load-saturated disks and the number of storage-saturated disks by N_L and N_S, respectively. It is easy to see that $R[1], \ldots, R[N_L]$ are load-saturated disks, and the rest are storage-saturated disks. Let m_j denote the number of colors assigned to disk d_j. Obviously for storage-saturated disks, $m_j = k$. Let c be the smallest fraction of load to which a storage-saturated disk is filled. Note that this disk must store a color with a number of items of that color being at most $c \times L/k$. (Minimum is at most the average.) Now every color on the unassigned list has no more than $c \times L/k$ remaining items of that color. (Otherwise, the Sliding-Window algorithm would have put this color on the lightest-loaded disk and increased greedily the total number of packed items.)

LEMMA 3.1. *Using the Sliding Window Algorithm, the number of unpacked items is at most* $\frac{c \times L \times N_L}{k}$.

Proof. The main thing we need to prove is that there are at most N_L colors left after we run the Sliding-Window algorithm. For each left over color we know that the number of items is at most $\frac{c \times L}{k}$, so the total number of unpacked items is at most $\frac{c \times L \times N_L}{k}$.

We examine the number of colors stored in the load-saturated disks. If there is a load-saturated disk d_j with $m_j < k$ colors, then there are no colors left when the algorithm terminates, i.e., all items are packed, which can be explained as follows. The reason that less than k colors are packed into d_j is due to the fact that at step j $\sum_{i=1}^{m_j} R[i] \geq L$. Since we sort the colors in a non-decreasing order, at this point any consecutive sequence of $k - 1$ colors in the list has the total size $\geq L$. Since at step j, we "split" at most one color[1], which is always added to the beginning of R, at any step $t \geq j$ we have a guarantee that, for the new list R, $\sum_{i=1}^{k} R[i] \geq L$, unless we have less than k colors. This implies that we fill the disks to their load capacity until we run out of colors. Hence we can pack all items.

We can now assume that all the load-saturated disks have k colors. The storage-saturated disks have k colors as well. We start with $M \leq N \times k$ colors. During the process, we can split at most N_L colors, i.e., we can generate at most N_L new "instances" of originally existing colors. This is because only filling disks that are load-saturated can result in generating new "instances" of colors. So the number of new "instances" of colors generated is upper bounded by the number of load-saturated disks. Thus the number of colors left is $\leq M + N_L - N \times k \leq N_L$. □

[1]By splitting a color we mean that, in the current iteration of the algorithm, only some of the items of that color are packed into the current disk; the remaining items might be packed in future iterations of the algorithm. Hence, this color might be packed into multiple disks.

COROLLARY 3.1. *For "uniform ratio" disks, if $\sum_{j=1}^{N} C_j \geq M + N - 1$, then all colors can be packed using the Sliding Window algorithm.*

Proof. This is an alternate proof for the claim in [11]. Our analysis of the algorithm makes the proof simpler.

Let $r = \frac{L_j}{C_j}$, denote the uniform ratio. Since the ratios' $\frac{L_j}{C_j}$ are uniform, once any disk becomes storage-saturated, the rest of the disks will be storage-saturated as well. The main claim we need to prove is that after we fill disk d_{N-1}, we have at most C_N colors left. We will prove this shortly. If d_{N-1} is storage-saturated, then we can safely assign the remaining C_N colors to d_N. If d_{N-1} is load-saturated, then all previous disks are load-saturated. Since the total number of items does not exceed the total load capacity, we will not exceed the load capacity.

We argue that if there is a load-saturated disk d_j with $m_j < C_j$, then all the items will be packed. At this stage, $R[\ell] \geq \frac{L_j}{m_j} \geq \frac{C_j}{C_j - 1} \times r \geq \frac{C_t}{C_t - 1} \times r$ for all $\ell > m_j$ and all $t \geq j$. Recall that disks are sorted in non-decreasing order of C_i. Thus we have the following result: at any step $t > j$, $\sum_{i=1}^{C_t} R[i] \geq \sum_{i=2}^{C_t} R[i] \geq C_t \times r = L_t$, and so all items are packed without sliding the window.

Since all load-saturated disks have C_j colors, after we fill disk d_{N-1}, we have generated at most $N-1$ new "instances" of colors. The total number of colors left is $\leq M + N - 1 - \sum_{i=1}^{N-1} C_i \leq C_N$. □

LEMMA 3.2. *Using the Sliding Window Algorithm, the number of unpacked items is at most $(1-c) \times L \times N_S$.*

Proof. At least $L \times N_L + c \times L \times N_S$ items are packed. Subtracting this quantity from an upper bound on the total number of items $N \times L$ gives $N_S \times L + N_L \times L - L \times N_L - c \times L \times N_S$, which yields the claim. □

THEOREM 3.1. *The Sliding Window Algorithm guarantees to pack $(1 - \frac{1}{(1+\sqrt{k})^2})$ fraction of items in the homogeneous case.*

Proof. The above two lemmas give us two upper bounds on the number of unpacked items. Hence the number of unpacked items is at most $\min(\frac{c \times L \times N_L}{k}, (1-c) \times L \times N_S)$. The number of packed items is at least $L \times N_L + c \times L \times N_S$. Hence the ratio of unpacked(U) to packed(S) items is at most

$$\frac{U}{S} \leq \frac{\min(\frac{c \times L \times N_L}{k}, (1-c) \times L \times N_S)}{L \times N_L + c \times L \times N_S}.$$

This yields

$$\frac{S}{U+S} \geq 1 - \frac{1}{(1+\sqrt{k})^2}$$

which proves the claim. (The details of this derivation are given in Appendix B.) □

This proof can also be extended to the uniform-ratio case. The motto in the homogeneous case is that the bigger the disk the better the performance of the Sliding Window Algorithm. So in a uniform-ratio system one should expect the algorithm to do at least as well as the homogeneous case where all disks assume the smallest disk size in the uniform-ratio system. The following theorem formally proves this intuition.

THEOREM 3.2. *The Sliding Window Algorithm guarantees to pack $(1 - \frac{1}{(1+\sqrt{C_{\min}})^2})$ fraction of items in the uniform ratio case where $C_{\min}$ denotes the minimum capacity of a disk in our system.*

Proof. Let $r = \frac{L_j}{C_j}$ for $j = 1 \ldots N$ denote the uniform ratio. From the proof of Corollary 3.1 above, if there is a load-saturated disk d_j with $m_j < C_j$, then all items will be packed. Thus we will focus on the case where for all j we have $m_j = C_j$. Let M_L denote the total number of colors (we count the same colors in different disks as different multiple colors) in the load-saturated disks $1, \ldots, N_L$, so $M_L = \sum_{j=1}^{N_L} m_j = \sum_{j=1}^{N_L} C_j$. Let M_S denote the total number of colors in the storage-saturated disks $N_L + 1, \ldots, N$, so $M_S = \sum_{j=N_L+1}^{N} m_j = \sum_{j=N_L+1}^{N} C_j$. Again let c be the smallest fraction of load to which a storage-saturated disk is filled. Thus we have the following similar results:

- For each left over color we know that the number of items is at most $c \times r$.
- There are at most N_L colors left unassigned. We have $N_L \leq \frac{M_L}{m_{min}} = \frac{M_L}{C_{\min}}$.
- The number of unpacked items is at most $c \times r \times \frac{M_L}{C_{\min}}$.
- At least $r \times M_L + c \times r \times M_S$ items are packed.
- The number of unpacked items is at most $r \times M_L + r \times M_S - r \times M_L - c \times r \times M_S = (1-c) \times r \times M_S$.

Hence the ratio of the unpacked(U) to packed(S) items is at most

$$\frac{U}{S} \leq \frac{\min(\frac{c \times r \times M_L}{C_{\min}}, (1-c) \times r \times M_S)}{r \times M_L + c \times r \times M_S}.$$

Let $y = \frac{M_L}{M_L + M_S}$ and thus $\frac{M_S}{M_L + M_S} = 1 - y$. Simplifying the upper bound for this expression, we obtain

$$\frac{\min(\frac{cy}{C_{\min}}, (1-c)(1-y))}{y + c(1-y)}.$$

Note that this is the same expression as in Theorem 3.1 for the homogeneous system. Optimizing this expression gives the same bound as in Equation 4.1 (Appendix B) with k replaced by $C_{\min}$. This proves the claim. □

3.1 Tight Example

We now give an example to show that the bound of $(1-\frac{1}{(1+\sqrt{k})^2})$ is tight. In other words, there are instances for which no solution will pack more than $(1-\frac{1}{(1+\sqrt{k})^2})$ fraction of items.

Assume that k, the storage capacity of a disk is a perfect square[2] and $k \geq 2$. (When k=1, the tight example is trivial.) Let N, the number of disks, be $1+\sqrt{k}$, and let $L = k+\sqrt{k}$. There are $\sqrt{k}$ colors with a large number of items each, $U_1, \ldots, U_{\sqrt{k}}$ with $|U_i| = 2+\sqrt{k}$ for $1 \leq i \leq \sqrt{k}$; we will refer to these as "large colors". And, there are $(k-1)(1+\sqrt{k})+1$ colors with a small number of items each, $U_{\sqrt{k}+1}, \ldots, U_{k(1+\sqrt{k})}$ with $|U_i| = 1$ for $\sqrt{k}+1 \leq i \leq k(1+\sqrt{k})$; we will refer to these as "small colors".

We will show that there are always at least $\sqrt{k}$ items that do not get packed. In this case, the fraction of items that are not packed is at least $\frac{\sqrt{k}}{(1+\sqrt{k})(k+\sqrt{k})}$ which is exactly $\frac{1}{(1+\sqrt{k})^2}$. This proves the claim.

We first consider the $\sqrt{k}$ large colors. An unsplit set U_i has all its items packed in a single disk. A split set U_i has its items packed in several disks. For a disk that contains at least one large unsplit color, the available load capacity left is at most $k-2$. (Note that after packing one large unsplit color, the available load capacity is smaller than the storage capacity.) We can exchange any of the remaining colors with $j \geq 2$ items of the same color, with any j small (distinct color) items in any other disk, while still packing the same number of items. These disks now have one large unsplit color, and at most $k-2$ small colors. The remaining disks have only large split colors. In fact, assume that there are exactly p $(0 \leq p \leq \sqrt{k})$ large colors that do not get split $U_1, \ldots, U_p$, with disk d_i containing U_i.

Now consider the remaining $N-p$ disks; we are left with at least $k \times N - p(k-1) = k \times (N-p) + p$ colors, but we only have $k \times (N-p)$ storage capacity left. Since the remaining $\sqrt{k}-p$ large colors are all split, this generates an additional $\sqrt{k}-p$ "instances" of colors. Thus we have at least $k \times (N-p) + p + \sqrt{k} - p$ colors. This will create an excess of $\sqrt{k}$ items that we cannot pack.

3.2 Numerical Results

We have implemented the Sliding Window algorithm and compared its performance to the theoretical results developed in this section. Refer to Figs. 1–3 for plots of the fraction of unassigned items given by our worst case bound (i.e., $\frac{1}{(1+\sqrt{k})^2}$), the bound corresponding to the corollary of the result in [11] (i.e., $\frac{1}{1+k}$), as well as the Sliding Window algorithm.

The results presented here are for the *homogeneous* case only. For the purposes of this comparison we generated the test cases using the Zipf distribution to determine the skewness in the number of items of each color. The Zipf distribution is defined as follows [9]:

$$\text{Prob[item of color } i] = \frac{c}{i^{(1-\theta)}} \qquad \forall\, i = 1, \ldots, M$$
$$\text{and} \quad 0 \leq \theta \leq 1$$
$$\text{where} \quad c = \frac{1}{H_M^{(1-\theta)}} \quad \text{and} \quad H_M^{(1-\theta)} = \sum_{j=1}^{M} \frac{1}{j^{(1-\theta)}}$$

where θ determines the degree of skewness. For instance, $\theta = 1.0$ corresponds to the uniform distribution whereas $\theta = 0.0$ corresponds to the skewness in access patterns often attributed to movies-on-demand type applications, e.g., similar to the *measurements* performed in [3].

We experimented with different values of θ and computed the percentage of items that can be packed by the Sliding Window algorithm as a function of k, the load capacity of a disk. The results of these experiments are given in Figs. 1–3. In all cases $L = 100$ and $N = 5$.

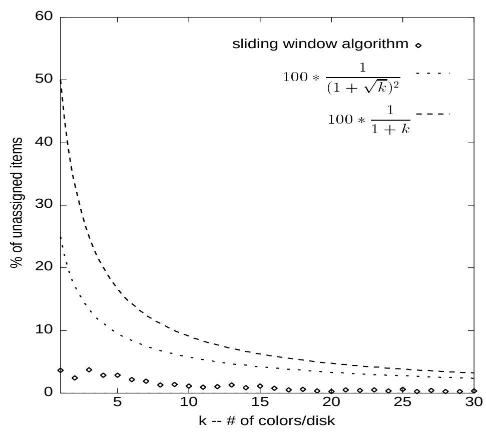

Figure 1: $\theta = 1.0$.

We can draw the following conclusions from these figures:

- the theoretical bound is reasonably tight when the the number of items of each color is fairly skewed (as in Figs. 2 and 3), as is the case in a VOD server, which is our motivational application; furthermore, the performance of the Sliding Window algorithm is very close to the theoretical bound for inputs which are "similar" to the tight example given in Section 3 (as in Fig. 2);

[2] When k is not a perfect square we can get arbitrarily close to the bound by modifying this family of examples.

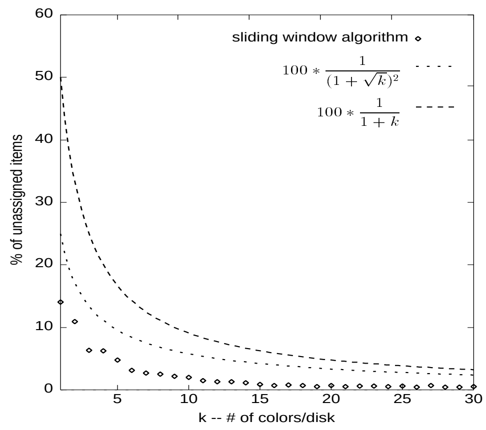

Figure 2: $\theta = 0.5$.

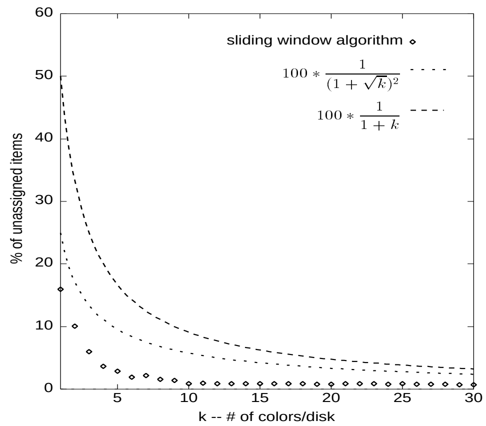

Figure 3: $\theta = 0.0$.

- the performance of the Sliding Window algorithm can be significantly better than the theoretical bound when the number of items of each color is approximately the same and each disk has a relatively small storage capacity (as in Fig. 1); however, the theoretical bound is reasonably tight for larger values of k (which again, is reasonable for our motivational application).

4 Polynomial Time Approximation Schemes

We now present an approximation scheme our data placement problem. We first present the approximation scheme for homogeneous storage systems and then briefly sketch the ideas used to get an approximation scheme for uniform ratio systems.

4.1 Homogeneous Storage Systems

We design an algorithm that for any fixed $\epsilon > 0$, gives a $(1-\epsilon)$-approximation in polynomial time. If the error parameter $\epsilon \geq 1/(1+\sqrt{k})^2$, then we can simply use the Sliding Window algorithm to obtain a $(1-\epsilon)$-approximation. In the rest of this section, we focus on the case when $\epsilon < 1/(1+\sqrt{k})^2$. In other words, k can be assumed to be a constant when ϵ is a fixed constant. Our approximation scheme involves the following steps:

1. First we show that any given input instance can be approximated by another instance I' such that no color class in I' contains "too many" items.
2. Next we show that for any input instance there exists a near-optimal solution that satisfies certain structural properties concerning how items are assigned to the disks.
3. Finally, we give an algorithm that in polynomial time finds the near-optimal solution referred to in step (2) above, provided that the input instance is as determined by step (1) above.

We now describe in detail each of these steps. In what follows, we use $\mathsf{OPT}(I)$ to denote an optimal solution to the instance I and α to denote $1/\epsilon$. Also, for any solution S, we use $|S|$ to denote the number of items packed by it.

4.1.1 Preprocessing the Input Instance

We say that an instance I is *B-bounded* if the size of each color class is at most B.

LEMMA 4.1. *For any instance I, we can construct in polynomial time another instance I' such that*

- *I' is (αL)-bounded,*
- *any solution S' to I' can be mapped to a solution S to I of identical value, and*
- $|\mathsf{OPT}(I')| \geq (1-\epsilon)|\mathsf{OPT}(I)|$.

Proof. Consider a color c_j in the instance I such that $|U_j| > \alpha L$. Replace c_j with a new set of colors $c_j^1, c_j^2, ..., c_j^s$ where $s = \lceil |U_j|/(\alpha L) \rceil$. Let U_j^i denote the set of items with color c_j^i where $1 \leq i \leq s$. Then $|U_j^1| = ... = |U_j^{s-1}| = \alpha L$ and $|U_j^s| = |U_j| - (s-1)\alpha L$. Repeat this procedure for any color class that has more than αL items in I. We now have our instance I'.

It is easy to see that any feasible solution to I' gives a feasible solution of same value to I; simply replace each color c_j^i with c_j.

Now consider a solution S for instance I. We show that it can be mapped to a solution S' of size $(1-\epsilon)|S|$ for I'. If $|U_j| \leq \alpha L$ for $1 \leq j \leq M$, then clearly S is also a feasible solution of the same value for I'. Otherwise, fix a color class U_j in I such that $|U_j| > \alpha L$. Label the occurrences of the items of color c_j as $1, 2, ...$ as we move from d_1 to d_N in solution S. Replace the ith occurrence of a color c_j item with an item of color c_j^l where $l = \lceil i/\alpha L \rceil$. The resulting solution may no longer be a feasible solution for I'. A disk may now contain items of two different colors, say c_j^l and c_j^{l+1}, in place of a single color c_j and hence the total number of colors in the disk may become $k+1$. We simply discard all the items in any disks where this event occurs. Repeat this procedure for every color class with more than αL items in I. We claim that we have discarded no more than an ϵ-fraction of packed items. The reason is that we throw away at most L items from a color class at a *crossover* disk but this event occurs only once in every αL occurrences of items packed from a color class. Thus what we discard is at most an ϵ-fraction of what is packed. □

4.1.2 Structured Approximate Solutions

Let us call a color class U_j *small* if $|U_j| \leq \epsilon L/k$, and *large* otherwise. Also, for a given solution, we say that a disk is *light* if it contains less than ϵL items, and it is called *heavy* otherwise. The lemma below shows that there exists a $(1-\epsilon)$-approximate solution where the interaction between light disks and large color classes, and between heavy disks and small color classes, obeys some nice properties.

LEMMA 4.2. *For any instance I, there exists a solution S satisfying the following properties:*

- *at most one light disk receives items from any large color class,*
- *a heavy disk is assigned either zero or all items in a small color class, and*
- *S packs at least $(1-\epsilon)\mathsf{OPT}(I)$ items.*

Proof. Let n_i denote the number of items assigned to the ith disk in the solution $\mathsf{OPT}(I)$. Relabel the disks 1 through N such that $n_1 \geq n_2 ... \geq n_N$. Assume w.l.o.g. that $\mathsf{OPT}(I)$ is a lexicographically maximal solution in the sense that among all optimal solutions, $\mathsf{OPT}(I)$ is one that maximizes the sums $\sum_{j=1}^{i} n_j$ for each $i \in [1..N]$.

It is easy to see that the first property follows from the maximal property of $\mathsf{OPT}(I)$. To establish that a heavy disk in $\mathsf{OPT}(I)$ receives either zero or all items from a small color class in the solution S, we may need to discard some items from the heavy disks in $\mathsf{OPT}(I)$. Let X be the set of heavy disks that contain at most $(1-\epsilon)L$ items from large color classes. Consider any disk $d_i \in X$ that receives some but not all items from a small color class U_j. Simply move all items of U_j to d_i. Repeat this process till no disk in X violates this property. Since a small color class has at most $\epsilon L/k$ items, clearly the capacity of no disk is violated in this process. Finally, for the remaining disks, simply discard any items from small color classes. Clearly, the resulting solution is $(1-\epsilon)$-approximate. □

For a given solution S, a disk is said to be δ-integral w.r.t. a color class U_j if it is assigned $\beta\lceil \delta L\rceil$ items from U_j, where $0 < \delta \leq 1$ and β is a non-negative integer.

LEMMA 4.3. *Any solution S can be transformed into a solution S' such that*

- *each heavy disk in S is (ϵ^2/k)-integral in S' w.r.t. each large color class, and*
- *S' packs at least $(1-\epsilon)|S|$ items.*

Proof. To obtain the solution S' from S, in each heavy disk, round down the number of items assigned from any heavy color class to the nearest integral multiple of $\lceil(\epsilon^2/k)L\rceil$. Then the total number of items discarded from any heavy disk in this process is at most

$$k\left(\left\lceil(\frac{\epsilon^2}{k})L\right\rceil - 1\right) \leq k\left((\frac{\epsilon^2}{k})L\right) \leq \epsilon(\epsilon L).$$

Since each heavy disk contains at least ϵL items, the total number of items discarded in this process can be bounded by $\epsilon|S|$. Thus S' satisfies both properties above. □

4.1.3 The Approximation Scheme

We start by preprocessing the given input instance I so as to create an (αL)-bounded instance I' as described in Lemma 4.1. We now give an algorithm to find a solution S to I' such that S satisfies the properties described in Lemmas 4.2 and 4.3 and packs the largest number of items. Clearly,

$$|S| \geq (1-\epsilon)^2|\mathsf{OPT}(I')| \geq (1-\epsilon)^3|\mathsf{OPT}(I)|.$$

Let O be an optimal solution to the instance I' that is lexicographically maximal. Assume w.l.o.g. that we know the number of heavy disks in O, say N'. Let $\mathcal{H}$ be the set of disks d_1 through $d_{N'}$ and let $\mathcal{L}$ be the remaining disks, $d_{N'+1}$ through d_N. The algorithm consists of two steps, corresponding to the packing of items in $\mathcal{H}$ and $\mathcal{L}$ respectively.

Packing items in $\mathcal{H}$: We first guess a vector $\langle l_1, l_2, ..., l_{N'}\rangle$ such that l_i denotes the number of small color classes to be assigned (completely) to a disk $d_i \in \mathcal{H}$. Since all disks are identical, we can guess such a vector in $O(N^{k+1})$ time by guessing a compact representation of the following form. We guess a vector $\langle q_0, q_1, \cdots, q_k\rangle$ such that $\sum_{i=0}^{k} q_i = N'$ and q_i denotes the number of disks in $\mathcal{H}$ that are assigned i small color classes (completely). It is easily seen that any such vector can be mapped to a vector of the form $\langle l_1, l_2, ..., l_{N'}\rangle$ and vice versa. Now proceeding from 1 through N', we assign to a disk d_i the largest size l_i small color classes that remain.

Next we use a dynamic program by moving across the disks from 1 through N' so as to find an optimal (ϵ^2/k)-integral solution for packing the largest number of items from the large color classes. For the purpose of this packing, the capacity of each heavy disk is restricted to be $(1-\epsilon)L$ and the number of color classes allowed in disk d_i is given by $k - l_i$. Let $\beta = k/\epsilon^3$ and $q = \lceil(\epsilon^2 L)/k\rceil$. The dynamic programming solution is based on maintaining a β-tuple $\vec{v} = \langle v_1, v_2, ..., v_\beta\rangle$ where v_i denotes the number of large color classes that have $i \cdot q$ elements available in them. Proceeding from $i = 1$ through N', we compute a table entry $T[\vec{v}, i]$ for each possible state vector $\vec{v}$. The entry indicates the largest number of items that can be packed in the disks d_1 through d_i subject to the constraint that the resulting state vector is $\vec{v}$. Since there are at most Nk color classes, the total number of state vectors is bounded by $(Nk)^{k/\epsilon^3}$, which is polynomial for any fixed ϵ.

Packing items in $\mathcal{L}$: We know that our solution need not assign items from a large color class to more than one disk in $\mathcal{L}$. Moreover, at most ϵL items from any large color class are packed in a disk in $\mathcal{L}$. So at this stage we can truncate down the size of each large color class to $\lfloor\epsilon L\rfloor$. We now construct an instance of the b-matching problem on a weighted bipartite graph $G = (X \cup Y, E)$ where X has a vertex for each disk in $\mathcal{L}$ and Y has a vertex for each remaining color class. For each $x \in X$ and $y \in Y$, there is an edge $(x, y) \in E$ whose weight is equal to the number of items remaining in the color class corresponding to y. Now we find a maximum weight matching such that the degree of each vertex in X is restricted to be k while the degree of each vertex in Y is restricted to be 1. Clearly, such a matching gives a feasible solution of maximum weight. This completes our approximation scheme.

4.2 Uniform Ratio Storage Systems

We now briefly sketch how the PTAS result above can also be extended to the case of uniform ratio storage systems. If $\epsilon \geq 1/(1+\sqrt{C_{\min}})^2$ then we can use the Sliding Window algorithm to obtain a $(1-\epsilon)$-approximation. On the other hand, if $C_{\min}$ as well as $C_{\max}$ are bounded by a constant (parameterized by ϵ), the approach of the preceding subsection easily extends to give a PTAS.

The difficulty thus lies in the case when $C_{\min}$ is small but $C_{\max}$ is relatively large. In other words, our system contains disks of widely varying storage capacities. We handle this case by showing that every "large" disk can be approximately represented by a collection of disks with bounded storage capacity such that we loose at most an ϵ-fraction of items due to this approximate representation. Once this transformation is made, we can once again use the approach of the preceding subsection to obtain a PTAS. We defer the details to the full version.

References

[1] S. Berson, S. Ghandeharizadeh, R. R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. *SIGMOD*, pages 79–90, 1994.

[2] C. Chekuri and S. Khanna. On multidimensional packing problems. In *ACM Symp. on Discrete Algorithms*, pages 185–194, 1999.

[3] A. L. Chervenak. Tertiary Storage: An Evaluation of New Applications. *Ph.D. Thesis, UC Berkeley*, 1994.

[4] C. F. Chou, L. Golubchik, and J. C.S. Lui. A performance study of dynamic replication techniques in continuous media servers. Technical Report CS-TR-3948, University of Maryland, October 1998.

[5] M. Dawande, J. Kalagnanam, and J. Sethuraman. Variable Sized Bin Packing With Color Constraints. Technical report, IBM Research Division, T.J. Watson Research Center, 1999.

[6] A. M. Frieze and M. R. B. Clarke. Approximation algorithms for the m-dimensional 0-1 knapsack problem: worst-case and probabilistic analyses. *European Journal of Operational Research*, pages 100–109, 1984.

[7] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco, 1979.

[8] S. Ghandeharizadeh and R. R. Muntz. Design and Implementation of Scalable Continuous Media Servers. *Parallel Computing Journal*, 24(1):91–122, January 1998.

[9] D. E. Knuth. *The Art of Computer Programming, Volume 3*. Addison-Wesley, 1973.

[10] P. Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, pages 130–143, 1988.

[11] H. Shachnai and T. Tamir. On two class-constrained versions of the multiple knapsack problem. *To appear in Algorithmica*.

[12] H. Shachnai and T. Tamir. Polynomial time approximation schemes for class-constrained packing problems. *Manuscript*, 1999.

[13] M. Stonebraker. A Case for Shared Nothing. *Database Engineering*, 9(1):4–9, 1986.

[14] J. Wolf, H. Shachnai, and P. Yu. DASD Dancing: A Disk Load Balancing Optimization Scheme for Video-on-Demand Computer Systems. In *ACM SIGMETRICS/Performance Conf.*, pages 157–166, 1995.

Appendix A: NP-hardness proof

In this appendix we give our proof of Theorem 1.3. We first define the PARTITION problem, and then describe a polynomial time reduction from it to the homogeneous data placement problem.

PARTITION: Given a finite set A, with a size $s(a)$ for each element $a \in A$, is there a subset $A' \subseteq A$ such that

$$\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)?$$

This remains NP-complete even if we require $|A'| = |A|/2$ [7]. Let $n = |A|$.

Proof. The problem is easily seen to be in NP, since a proposed solution can be trivially checked in polynomial time.

We now show a polynomial time reduction from PARTITION. Let $a_{\max}$ be the maximum size item. For the reduction, define $K = n \times C \times s(a_{\max})$ where C is a large constant. Let $L = \frac{3}{2}K$ and $N = n$. So there are n disks with $k = 2$ (storage capacity) and a large L value. Let $D = \sum_{i=1}^{n}(K + s(a_i)) = n \times K + \sum_{i=1}^{n} s(a_i)$. Let $M = n + 2$, with $x_i = \frac{1}{2}K - s(a_i)$ for $1 \leq i \leq n$ and $x_{n+1} = \frac{1}{2}D$ and $x_{n+2} = \frac{1}{2}D$.

Note that $\sum_{i=1}^{n+2} x_i = \sum_{i=1}^{n} x_i + D = \frac{3}{2}n \times K$ which is exactly NL, the storage capacity.

We claim that if there is a solution to the partition problem with $|A'| = |A|/2$ and with $s(A') = s(A - A')$ then there is a way to pack all items into the N disks. The items are packed as follows. Put x_i items of color i in disk i. Disk i now has space for one new color, and exactly $\frac{3}{2}K - (\frac{1}{2}K - s(a_i))$ items. This is exactly $K + s(a_i)$. If item $a_i \in A'$ then add items of color $n+1$ to disk i, otherwise add items of color $n+2$ to disk i. Since each disk can hold two colors, this does not violate the color. Note that the number of items of color $n+1$ that we pack is exactly $\sum_{a_i \in A'}(K + s(a_i)) = \frac{n}{2} \times K + s(A') = \frac{1}{2}D$. The calculation is identical for items of color $n+2$, and this concludes the proof that all items are packed.

We now argue that if there is a solution to 2-HDP where all items get packed, then there is a solution to the PARTITION problem. We first claim that if all items are packed, then items of colors i and j, with $1 \leq i, j \leq n$ cannot be packed into the same disk. This is the case since: (a) only two colors can be packed in a disk and hence no other color can go into that disk, and (b) the total capacity used up by items of color i and j, $1 \leq i, j \leq n$, would not exceed K, which is much less than the capacity of the disk. If we cannot saturate the disk to full load capacity, then we cannot pack all items (since the number of items exactly equals the total load capacity). If each items of color i is in a distinct disk, then without loss of generality we pack items of color i in disk i and now we are left with items of only two colors that we need to split equally between the disks, and each disk can only take an item of one color, with $K + s(a_i)$ items of that color. Since $K >> s(a_i)$ we must pack items of color $n+1$ in exactly $n/2$ disks, and the items of color $n+2$ in the remaining $n/2$ disks. Let $A' = \{a_i | \text{items of color } n+1 \text{ are packed in disk } i\}$. As said earlier $|A'| = \frac{n}{2}$, and $s(A') = s(A - A')$. This completes the proof. $\square$

When $k \geq 3$, the problem can be seen to be strongly NP-hard for even the homogeneous case by a simple reduction from 3-PARTITION [7].

Appendix B: Details of Proof of Theorem 3.1

Given that the ratio of unpacked(U) to packed(S) items is at most

$$\frac{U}{S} \leq \frac{\min(\frac{c \times L \times N_L}{k}, (1-c) \times L \times N_S)}{L \times N_L + c \times L \times N_S}$$

let $y = \frac{N_L}{N}$ and thus $\frac{N_S}{N} = 1 - y$. Simplifying the upper bound for the number of unpacked to packed items, we obtain

$$\frac{\min(\frac{cy}{k}, (1-c)(1-y))}{y + c(1-y)}. \tag{4.1}$$

This is the same as

$$\min(\frac{\frac{cy}{k}}{y + c(1-y)}, \frac{(1-c)(1-y)}{y + c(1-y)}).$$

We can simplify the two functions to the following

$$\min(\frac{1}{k \times (\frac{1}{c} + \frac{1-y}{y})}, \frac{(1-c)(1-y)}{1 - (1-c)(1-y)}).$$

The first term is strictly increasing as c or y increases, while the second term is strictly decreasing as c or y increases. So in order to maximize the expression, we need to set the two terms equal, which means

$$\frac{cy}{k} = (1-c)(1-y).$$

This gives

$$y = \frac{1-c}{1-c+\frac{c}{k}}.$$

Substituting for y gives us that the upper bound for this ratio is at most

$$\frac{c - c^2}{k - kc + c^2}.$$

This achieves its maxima when $c = (1 - \frac{1}{1+\sqrt{k}})$. The fraction of all items that are packed is

$$\frac{S}{U+S} = \frac{1}{1 + \frac{U}{S}}.$$

Replacing the bound that we derived for c we get that

$$\frac{U}{S} \leq \frac{1}{k + 2\sqrt{k}}.$$

This yields

$$\frac{S}{U+S} \geq 1 - \frac{1}{(1+\sqrt{k})^2}.$$