The Journal of Systems and Software 66 (2003) 157–166

The Journal of Systems and Software

www.elsevier.com/locate/jss

Variable-size data item placement for load and storage balancing

Yung-Cheng Ma [a,*], Jih-Ching Chiu [b], Tien-Fu Chen [c], Chung-Ping Chung [a]

[a] *Department of Computer Science and Information Engineering, National Chiao-Tung University, 1001 Ta Hsueh Road, Hsinchu 30050, Taiwan*

[b] *Department of Electrical Engineering, National Sun-Yat Sun University, Kaohsiung 80424, Taiwan*

[c] *Department of Computer Science and Information Engineering, National Chung-Cheng University, Chayi 600, Taiwan*

Received 10 May 2001; received in revised form 19 October 2001; accepted 14 November 2001

Abstract

The rapid growth of Internet brings the need for a low cost high performance file system. Two objectives are to be pursued in building such a large scale storage system on multiple disks: load balancing and storage minimization. We investigate the optimization problem of placing variable-size data items onto multiple disks with replication to achieve the two objectives. An approximate algorithm, called *LSB_Placement*, is proposed for the optimization problem. The algorithm performs bin packing along with MMPacking to obtain a load balanced placement with near-optimal storage balancing. The key issue in deriving the algorithm is to find the optimal bin capacity for the bin packing to reduce storage cost. We derive the optimal bin capacity and prove that *LSB_Placement* algorithm is asymptotically 1-optimal on storage balancing. That is, when the problem size exceeds certain threshold, the algorithm generates a load balanced placement in which the data sizes allocated on disks are almost balanced. We demonstrate that, for various Web applications, a load balanced placement can be generated with disk capacity not exceeding 10% more than the balanced storage space. This shows that the *LSB_Placement* algorithm is useful in constructing a low cost and high performance storage system.

1. Introduction

This paper deals with challenges in designing a large scale storage system for Web applications. A Web server stores large amount of data on multiple disks. In response to tremendous data retrieving requests, data items (files) should be distributed to balance workload for all disks, and frequently accessed items may be replicated on multiple disks. Assuming that the capacity of each disk is identical, amount of data allocated to all disks should be kept balanced such that the storage efficiency of these disks can be maximized. We investigate such a data placement problem for load and storage balancing.

Data placement has been widely studied (Johnson et al., 1974; Dowdy and Foster, 1982; Wah, 1984; Wolf and Pattipati, 1990; Rotem et al., 1993; Lee and Park, 1995; Little and Venkatesh, 1995; Narendran et al., 1997; Serpanos et al., 1998; Lee et al., 2000). In early 1970s, researchers investigated data placement for minimizing the storage cost (Johnson et al., 1974). From 1980s, needs in high performance database systems has turned the research focus to improve the data retrieval performance (Dowdy and Foster, 1982; Wah, 1984; Wolf and Pattipati, 1990; Rotem et al., 1993; Lee and Park, 1995; Little and Venkatesh, 1995; Narendran et al., 1997; Lee et al., 2000). Starting in late 1990s, information explosion brought by the Internet raises new challenges in designing storage systems—both performance and storage cost have to be taken into consideration. Serpanos et al. (1998) proposed the MMPacking algorithm, which distributes identical-size data items onto multiple disks for both load and storage balancing. In this paper, we extend the work of MMPacking and propose an algorithm which places variable-size data items to generate a load and storage balanced placement.

The optimization problem is as follows. The input is a set of data items, each associated with an access probability and a data size; and a set of (identical) disks. The output is a placement that places data items onto disks in which frequently accessed data items may be

[*] Corresponding author. Tel.: +886-3-5712121; fax: +886-3-5724176.
E-mail address: ycma@csie.nctu.edu.tw (Y.-C. Ma).

replicated to multiple disks. The objective is to find a load balanced placement which also minimizes required capacity for each disk.

We propose an approximate algorithm for the optimization problem. The algorithm first performs bin packing Horowitz et al., 1996 to pack data items into a set of bins with approximately equal sizes. This reduces the variable-size data item placement to the identical-size data item placement, which can be dealt with by MMPacking (Serpanos et al., 1998). The key issue is to find an optimal bin capacity for bin packing to minimize the required disk capacity. Choosing a large bin capacity results in uniform packed bin size but increases the replication overhead. The trade-off is modeled as a capacity function, which maps the bin capacity to the required disk capacity. The optimal bin capacity can then be calculated.

The performance of the proposed algorithm is analyzed mathematically. We prove that the proposed algorithm generates a load balanced placement with asymptotically 1-optimal storage cost. The solution found by the proposed algorithm approaches optimal storage balancing when the problem size exceeds a certain threshold. Statistics shows that, for most Web applications, a load balanced placement can be obtained with disk capacity not exceeding 10% more than the balanced storage space. This shows that the proposed algorithm is effective in constructing a low cost and high performance storage system.

This paper is outlined as follows. Section 2 formulates the optimization problem. The metric for evaluating an approximate algorithm and two related optimization problems are introduced in Section 3. Section 4 gives a brief description on the proposed algorithm. We show that the proposed algorithm produces a load balanced placement in Section 5. Section 6 analyzes the storage requirement to derive the optimal bin capacity. Section 7 completes the proposed algorithm and shows the asymptotic behavior of the algorithm. The usefulness of applying the proposed algorithm to real world applications is demonstrated in Section 8. Finally, a conclusion is given in Section 9.

2. Problem modeling

The optimization problem can be modeled as follows. The input is a set of N data items $I = \{I_0, I_1, \dots, I_{N-1}\}$ and a set of M disks $D = \{D_0, D_1, \dots, D_{M-1}\}$. Each data item I_i is associated with an access probability p_i and a data size s_i. The data sizes are normalized such that the size of the largest item is 1.00 and $0 < s_i \leqslant 1.00$ for any data item I_i. Items in I are to be placed on disks in D, and frequently accessed items are allowed to be replicated to multiple disks. The optimization problem is to find a placement such that minimum single disk capacity is required and the access loads to all disks are also balanced.

A *placement* can be represented by an $N * M$ matrix X, in which a row corresponds to a data item and a column corresponds to a disk. An entry at row i and column k, denoted X_{ik}, represents the ratio of access probability of item I_i imposed on disk D_k. The following properties hold:

$$0 \leqslant X_{ik} \leqslant 1 \text{ and} \tag{1}$$

$$\sum_{k=0}^{M-1} X_{ik} = 1 \quad \text{for each } I_i \tag{2}$$

Distributing the access probability of an I_i across multiple disks is to replicate the item onto multiple disks. A copy of I_i is to be placed on disk D_k if $X_{ik} > 0$. For a placement X, the required storage on disk D_k is thus

$$\text{size}_X(D_k) = \sum_{i=0}^{N-1} s_i * \lceil X_{ik} \rceil \tag{3}$$

where $\lceil X_{ik} \rceil = 1$ if $X_{ik} > 0$ and $\lceil X_{ik} \rceil = 0$ if $X_{ik} = 0$.

The data request load is formulated as follows. We assume that the load of accessing a data item once is proportional to the data size of the item. The load to the server by I_i is $p_i * s_i$. And the aggregate load L of the server

$$L = \sum_{i=0}^{N-1} p_i * s_i \tag{4}$$

For a placement X, the load share of accessing I_i on disk D_k is $X_{ik} * p_i * s_i$, and the load of a disk D_k is

$$\text{Load}_X(D_k) = \sum_{i=0}^{N-1} X_{ik} * p_i * s_i \tag{5}$$

A *load balanced placement* is a placement X in which the load of each disk equals the balanced load L/M.

$$\text{Load}_X(D_k) = \sum_{i=0}^{N-1} X_{ik} * p_i * s_i = \frac{L}{M} \quad \text{for each disk } D_k \tag{6}$$

The cost of a placement X is the largest size of the disks

$$\text{cost}(X) = \max_{D_k}\{\text{size}_X(D_k)\} = \max_{D_k}\left\{ \sum_{i=0}^{N-1} s_i * \lceil X_{ik} \rceil \right\} \tag{7}$$

The objective of this research is to find a placement X to minimize Eq. (7) subject to Eqs. (1), (2) and (6).

3. Background and related work

We follow Johnson et al. (1974) to define the metric to evaluate an approximate algorithm of an optimization problem (minimizing certain cost). Given a problem instance J, we denote $F(J)$ the cost of the solution found by an approximate algorithm and $\mathrm{OPT}(J)$ the cost of the optimal solution for the problem instance J. In this study, a problem instance represents a pair (I, D) of data item set I and disk set D. The performance of an approximate algorithm is indicated by the upper bound on the ratio $F(J)/\mathrm{OPT}(J)$ for all problem instance J. What we concern is the asymptotic behavior when the problem size (and hence $\mathrm{OPT}(J)$) is large. By writing the upper bound on $F(J)/\mathrm{OPT}(J)$ as a function $G(\mathrm{OPT}(J))$,

$$\frac{F(J)}{\mathrm{OPT}(J)} \leqslant G(\mathrm{OPT}(J)) \tag{8}$$

we call that an approximate algorithm is ϵ—optimal if

$$\lim_{\mathrm{OPT}(J)\to\infty} G(\mathrm{OPT}(J)) = \epsilon \tag{9}$$

This ϵ is the metric to be used to evaluate our proposed algorithm.

We introduce two related optimization problems and corresponding approximate algorithms. Our proposed algorithm uses these algorithms as building blocks.

3.1. MMPacking for identical-size data item placement

Serpanos et al. (1998) proposed the MMPacking algorithm for placing data items with identical sizes. The input is a set of n objects $B = \{B_0, B_1, \ldots, B_{n-1}\}$ with identical data sizes, and a set of M disks $D = \{D_0, D_1, \ldots, D_{M-1}\}$. Each object B_j is associated with an estimated load to access B_j, denoted $\mathrm{Load}(B_j)$, according to the access probability. MMPacking algorithm places objects in B onto disks in D with frequently accessed objects been replicated. Serpanos et al. (1998) has proved the following properties for the MMPacking algorithm.

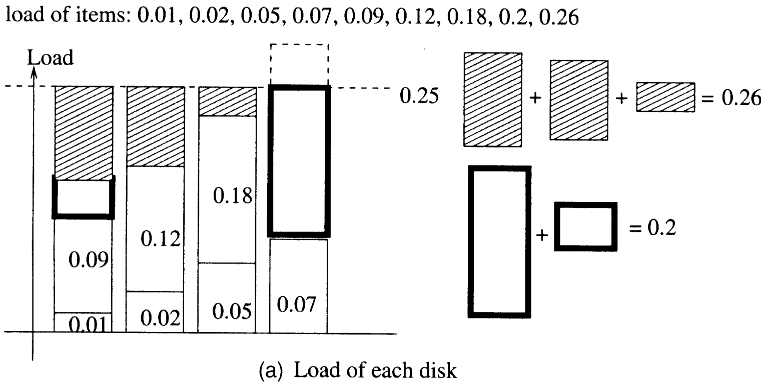
Property 1. *The MMPacking algorithm generates a load balanced placement.*

Property 2. *The MMPacking algorithm places at least $\lfloor n/M \rfloor$ and at most $\lceil n/M \rceil + 1$ objects on each disk.*

Fig. 1 illustrates the operations of MMPacking. Objects are sorted in increasing order of load and then assigned to disks in round-robin. Once the accumulated load of a disk exceeds the balanced load, the load of the object is split to the next disk in round-robin again. Splitting the load of an object is to replicate the object to multiple disks. The object with load 0.2 is replicated to the fourth and the first disk. Similar to the X matrix in Section 2, we represent the output of MMPacking with an $n * M$ matrix Y. For the example in Fig. 1, the matrix Y is shown in Fig. 2.

3.2. Bin packing

The bin packing problem Horowitz et al., 1996 is as follows. The input is a set of items $I = \{I_0, I_1, \ldots, I_{N-1}\}$



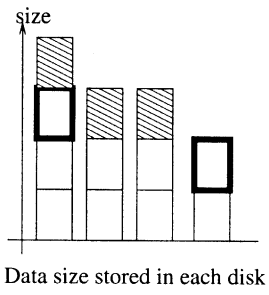


Fig. 1. A placement example using the MMPacking algorithm.

$$\begin{array}{c|cccc|}
 & D_0 & D_1 & D_2 & D_3 \\
B_0 & 1.00 & 0.00 & 0.00 & 0.00 \\
B_1 & 0.00 & 1.00 & 0.00 & 0.00 \\
B_2 & 0.00 & 0.00 & 1.00 & 0.00 \\
B_3 & 0.00 & 0.00 & 0.00 & 1.00 \\
B_4 & 1.00 & 0.00 & 0.00 & 0.00 \\
B_5 & 0.00 & 1.00 & 0.00 & 0.00 \\
B_6 & 0.00 & 0.00 & 1.00 & 0.00 \\
B_7 & 0.10 & 0.00 & 0.00 & 0.90 \\
B_8 & 0.50 & 0.42 & 0.08 & 0.00
\end{array}$$

Fig. 2. Result of MMPacking of the previous example.

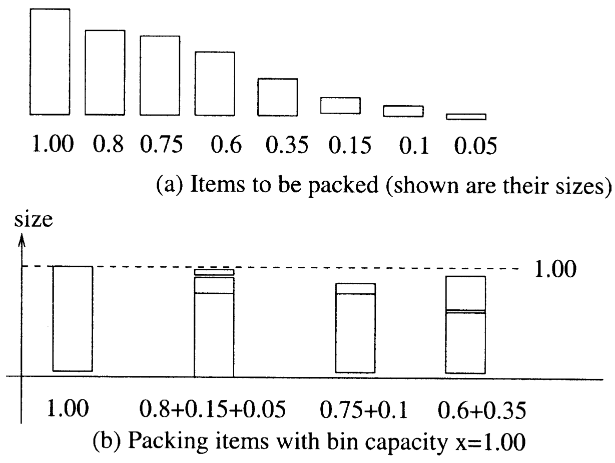

1.00
0.8
0.75
0.6
0.35
0.15
0.1
0.05
(a) Items to be packed (shown are their sizes)
size
1.00
1.00
0.8+0.15+0.05
0.75+0.1
0.6+0.35
(b) Packing items with bin capacity x=1.00

Fig. 3. Example of bin-packing.

and a bin capacity x. Each item I_i is associated with a size s_i of it. The goal is to pack the set of items I into the minimum number of bins $B = \{B_0, B_1, \dots, B_{n-1}\}$. Fig. 3 depicts an example of packing items with size not exceeding 1.00 to a set of bins with capacity $x = 1.00$.

We use the best-fit algorithm Horowitz et al., 1996 to perform bin-packing. This algorithm iteratively places an item to a bin with the smallest room left. The bin-packing property which we will use in our design is

Property 3. *During the best-fit bin-packing process, a new bin is initialized only when the current item to be placed cannot be fit in any existing bin.*

4. Framework of the approximate algorithm

We find that MMPacking and bin-packing can be useful in dealing with the load and storage balanced placement for variable-size data items. An almost complete description of the approximate algorithm is as follows:

Phase 1: Perform best-fit bin-packing to pack items in I into a set of bins $B = \{B_0, B_1, \dots, B_{n-1}\}$ with bin capacity x.

Phase 2: Perform MMPacking to obtain a placement Y for bins in B on disks in D. The load of B_j is the sum of the loads of all I_is in B_j.

$$\mathrm{Load}(B_j) = \sum_{I_i \in B_j} p_i * s_i \tag{10}$$

Phase 3: For each item I_i, $X_{ik} \leftarrow Y_{jk}$ where B_j is the bin containing I_i.

The key idea is problem reduction. The first phase performs bin packing to pack items into a set of bins with approximately equal storage requirements. The problem is thus reduced to the identical-size data item placement that can be dealt with by MMPacking.

The algorithm described above is almost complete. Only the bin capacity x for Phase 1 is not given here. We first show that the algorithm generates a load balanced placement in Section 5. Section 6 establishes the relationship between the bin capacity x and the disk capacity required to find the optimal bin capacity. Section 7 gives the complete description of the algorithm and summarizes its behavior on different problem sizes.

5. Load balancing of proposed algorithm

It is a direct consequence of Property 1 that the proposed algorithm can generate a load balanced placement. Imagine that a block in Fig. 1(a) represents load of a bin shared by a disk. (Disk D_k sharing the load of a bin B_j means that each item $I_i \in B_j$ is stored a copy in D_k with access probability X_{ik} shared.) Items in B_j contribute load $\sum_{I_i \in B_j} X_{ik} * p_i * s_i$ to D_k. This value is the load of B_j shared by D_k. A load balanced placement is obtained if the load of a bin is shared as expected in the MMPacking. This is imposed by setting $X_{ik} = Y_{jk}$ for each $I_i \in B_j$ (cf. Phase 3 of the proposed algorithm).

Theorem 1 (Load balancing). *The proposed algorithm generates a load balanced placement.*

Proof. Let X be the output of the proposed algorithm and Y be the intermediate result produced by MMPacking. Load of a disk D_k is the total load of all bins shared on D_k.

$$\begin{aligned}\mathrm{Load}_X(D_k) &= \sum_{i=0}^{N-1} X_{ik} * p_i * s_i \\ &= \sum_{B_j : Y_{jk} \neq 0} \sum_{I_i \in B_j} X_{ik} * p_i * s_i \end{aligned} \tag{11}$$

Since $X_{ik} = Y_{jk}$ for each $I_i \in B_j$ (cf. Phase 3 of the proposed algorithm), we have

$$\sum_{I_i \in B_j} X_{ik} * p_i * s_i = Y_{jk} * \sum_{I_i \in B_j} p_i * s_i = Y_{jk} * \text{Load}(B_j) \tag{12}$$

and hence

$$\text{Load}_X(D_k) = \sum_{B_j : Y_{jk} \neq 0} Y_{jk} * \text{Load}(B_j) \tag{13}$$

According to Property 1, MMPacking generates a load balanced placement:

$$\begin{aligned} \text{Load}_Y(D_k) &= \sum_{B_j : Y_{jk} \neq 0} Y_{jk} * \text{Load}(B_j) \\ &= \frac{L}{M} \quad \text{for each disk } D_k \end{aligned} \tag{14}$$

where L is the total load of all objects and M is the number of disks. Hence we have $\text{Load}_X(D_k) = L/M$. This proves the theorem. □

6. Setting bin capacity for minimizing storage cost

This section analyzes the choice of bin capacity, with which we can determine the disk capacity for a load balanced placement. In Section 6.1, we consider the case when the bin capacity is the size of the largest item. In Section 6.2, we allow the bin capacity to be larger than the size of the largest item for better storage balancing. We omit the subscript X in the notation $\text{size}_X(D_k)$ since only the placement generated by the proposed algorithm is considered.

6.1. Case of elementary bin capacity

We first consider the case of setting bin capacity $x = 1$, the size of the largest item, for bin packing. We prove that bin packing generates a set of bins with sizes exceeding 1/2 except for the smallest bin (Lemma 1). The number of bins generated are bounded (Corollary 1), and the required disk capacity is obtained (Theorem 2).

Lemma 1. *With bin capacity $x = 1$, there is at most one bin filled with size less than 1/2 in the output of the bin-packing.*

Proof. We prove this by induction on the number of items placed. The induction hypothesis is the lemma itself. The basis, after placement of item I_0, is trivial. Suppose the lemma holds after placement of I_i. The lemma holds again after placement of I_{i+1} if no new bin is initialized for I_{i+1}. We focus on the case that a new bin is initialized to place I_{i+1}. If size of each bin is at least 1/2 after placement of I_i (see Fig. 4(a)), the new bin is the only possible one with size not exceeding 1/2 after placement of I_{i+1}. In case that there is a unique bin B_j with size less than 1/2 after placement of I_i (see Fig. 4(b)), according to Property 3, each of the initialized bin has no room for I_{i+1} and hence size $S_{i+1} \geqslant 1/2$. The size of the new bin will exceed 1/2 and the bin B_j is still the only one with size not exceeding 1/2 after placement of I_{i+1}. The lemma holds again after placement of I_{i+1} for all possible cases. □

Let S be the total data size of all items,

$$S = \sum_{i=1}^{N} s_i \tag{15}$$

We derive the upper bound on the number of bins generated.

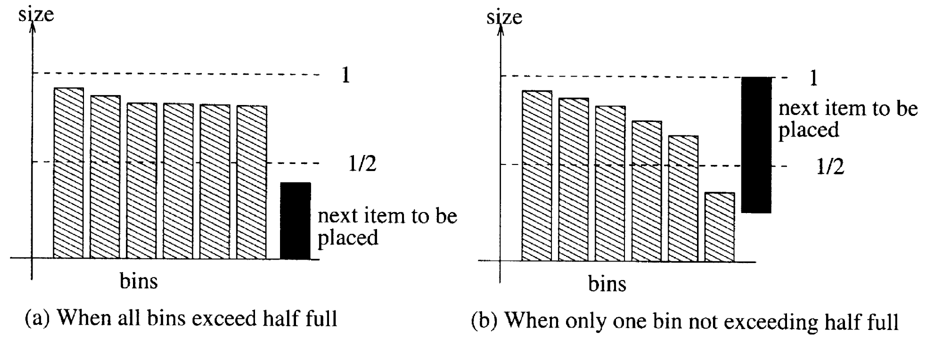
Corollary 1. *With bin capacity $x = 1$, the number of bins n generated by the bin-packing phase is bounded as follows:*

$$n \leqslant 2 * S + 1 \tag{16}$$

Proof. According to Lemma 1, the total size of all items is at least the total data size of the $(n - 1)$ bins with size exceeding 1/2.

$$s \geqslant \frac{1}{2} * (n - 1) \tag{17}$$

Eq. (16) is thus obtained. □



(a) When all bins exceed half full (b) When only one bin not exceeding half full

Fig. 4. Initializing a new bin to place an item.

Let M be the number of disks. We derive the required disk capacity as follows:

Theorem 2 (Required disk capacity). *With bin capacity $x = 1$, the proposed algorithm generates a placement in which*

$$size(D_k) \leqslant 2 * \frac{S}{M} + 3 \tag{18}$$

for each disk D_k.

Proof. MMPacking places at most $\lceil n/M \rceil + 1$ bins in each disk (Property 2) and the data size of each bin is at most 1.00. Hence we have the upper bound on the data size allocated on disk D_k:

$$\text{size}(D_k) \leqslant \lceil n/M \rceil + 1 \tag{19}$$

Total number of bins n is bounded from above as stated in Eq. (16). We thus obtain Eq. (18). □

This theorem states that a load balanced placement can be generated by the algorithm if the capacity of each disk exceeds two times the balanced data size (S/M) plus some constant (three times the size of the largest item).

6.2. Case of enlarged bin capacity

We improve storage balancing by enlarging bin capacity. Fig. 5 depicts an example of no items being packed together if bin capacity is set to be 1. (Recall that size is normalized such that the size of the largest item is 1.00 and size $s_i \leqslant 1$ for any item I_i.) In this example, some items are with size 1.00 and some are with size slightly greater than 1/2. In the worst case, the size of the disk containing most data approaches twice the data size of the disk containing least data. However, the worst case can easily be improved by enlarging bin capacity. The key issue is to find the optimal bin capacity to minimize required disk capacity.

Selecting bin capacity encounters a trade-off. Suppose the bin capacity is set to be $x > 1.00$. Let n be the number of bins generated and M be the number of disks. Fig. 6 depicts maximum difference on allocated data size between disks. MMPacking Serpanos et al., 1998 allocates $\lfloor n/M \rfloor$ to $\lfloor n/M \rfloor + 2$ bins on each disk. Except for the smallest bin, the size of each bin generated by bin-packing lies between $x - 1$ and x (Lemma 2). Data size allocated on a disk is bounded from above as follows:

$$\max_{D_k}\{\text{size}(D_k)\} \leqslant \left(\left\lfloor \frac{n}{M} \right\rfloor + 2\right) * x$$

In a disk, there is at most one bin with size not exceeding $x - 1$. Data size allocated on a disk is bounded from below as follows:

$$\min_{D_k}\{\text{size}(D_k)\} \geqslant \left(\left\lfloor \frac{n}{M} \right\rfloor - 1\right) * (x - 1)$$

Difference on allocated data size between disks is as follows:

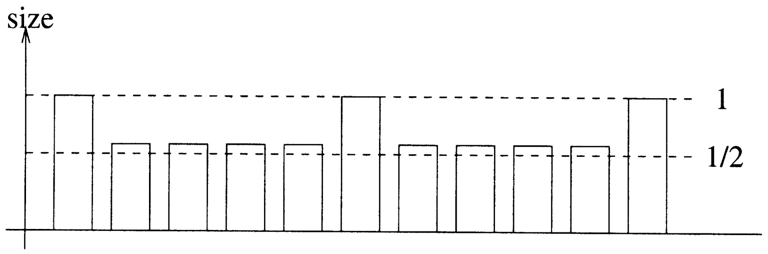


Fig. 5. The worst case of setting bin capacity to be the size of the largest item.

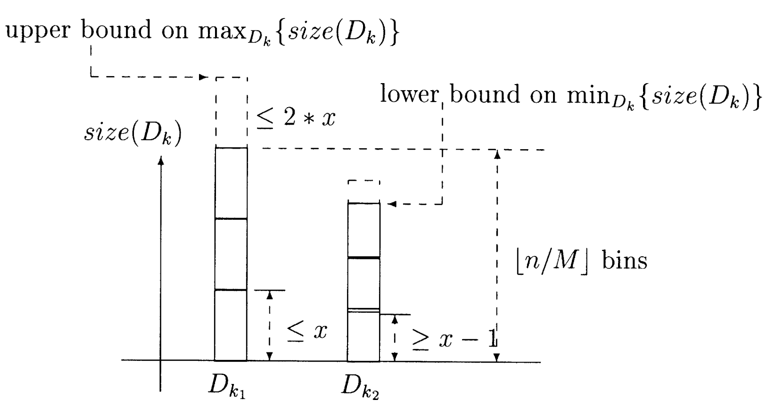

upper bound on $\max_{D_k}\{size(D_k)\}$
$\leq 2 * x$
$size(D_k)$
lower bound on $\min_{D_k}\{size(D_k)\}$
$\lfloor n/M \rfloor$ bins
$\leq x$
$\geq x - 1$
D_{k_1}
D_{k_2}

Fig. 6. Effects of bin capacity selection.

$$\max_{D_k}\{\text{size}(D_k)\} - \min_{D_k}\{\text{size}(D_k)\} = \mathrm{O}\left(\frac{n}{M}\right) + \mathrm{O}(x)$$

Selecting a large x reduces number of bins n generated and hence reduces $\mathrm{O}(n/M)$ in the above equation. However, selecting a large x increases $\mathrm{O}(x)$ in the above equation. We resolved the trade-off analytically.

Derivation of bin capacity and storage requirement is outlined as follows. Lemma 2 states packed bin sizes. Corollary 2 bounds a number of generated bins according to packed bin sizes. With the bound on number of generated bins, Lemma 3 relates storage requirement to selected bin capacity and defines the *capacity function* (Eq. (24)). The optimal bin capacity x is selected to minimize the capacity function. The storage requirement for a disk can also be derived (Theorem 3).

Lemma 2. *With bin capacity $x > 1$, there will be at most one bin filled with size less than $x - 1$ in the output of the bin packing.*

Proof. We prove this lemma by induction on the number of items placed. The induction hypothesis is the lemma itself. The basis, status after placing item I_0, is trivial. Suppose the lemma holds after placement of I_i. There are two cases for the status after placement of I_i: (i) size of each bin exceeds $x - 1$ (Fig. 7(a)), and (ii) there is a unique bin B_j with size not exceeding $x - 1$ (Fig. 7(b)). For case (i), the new bin (if initialized) is the only possible one with size less than $x - 1$ after placement of I_{i+1}. For case (ii), no new bin will be initialized (Property 3) since size $s_{i+1} \leqslant 1$ and at least B_j has room for I_{i+1}. B_j is the only possible bin with size not exceeding $x - 1$ after placement of I_{i+1}. The lemma holds again after placement of I_{i+1}. □

Let S be the total data size of all items. Similar to Corollary 1, we derive the upper bound on the number of bins generated.

Corollary 2. *With bin capacity $x > 1$, the number of bins n generated by bin packing is bounded as follows*:

$$n \leqslant \frac{S}{x-1} + 1 \tag{20}$$

Proof. According to Lemma 2, there are at least $n - 1$ bins with sizes exceeding $x - 1$, and the total data size S exceeds the total size of these $n - 1$ bins,

$$S \geqslant (x-1) * (n-1) \tag{21}$$

Eq. (20) is obtained immediately. □

Lemma 3. *With bin capacity $x > 1$, the proposed algorithm generates a placement in which*

$$\text{size}(D_k) \leqslant \frac{S}{M} * \left(1 + \frac{1}{x-1}\right) + 3x \tag{22}$$

for each disk D_k.

Proof. In the output of the proposed algorithm, each disk D_k contains at most $\lceil n/M \rceil + 1$ bins with the sizes of all bins not exceeding x.

$$\text{size}(D_k) \leqslant (\lceil n/M \rceil + 1) * x \tag{23}$$

Corollary 2 gives an upper bound on n and Eq. (22) is obtained immediately. □

We define the capacity function to indicate the required capacity of a disk if bin capacity is set to be x.

$$f(x) \equiv \frac{S}{M} * \left(1 + \frac{1}{x-1}\right) + 3x \tag{24}$$

The curve of the capacity function on the x–y plane is depicted in Fig. 8, which reflects the trade-off on selecting the bin capacity. Taking differential on $f(x)$ and solving the equation $f'(x) = 0$, we obtain the *optimal bin capacity* x_0 to minimize $f(x)$:

$$x_0 = 1 + \sqrt{\frac{S}{3 * M}} \tag{25}$$

And the required disk capacity for a load balanced placement is obtained.

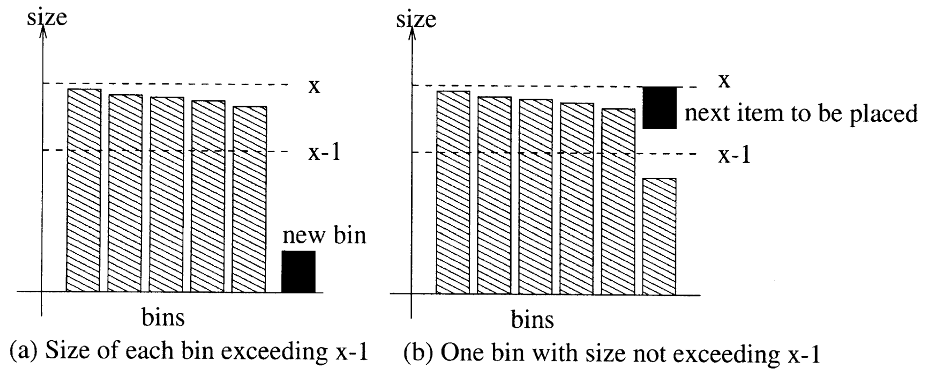

size
x
x-1
new bin
bins
(a) Size of each bin exceeding x-1
size
x
next item to be placed
x-1
bins
(b) One bin with size not exceeding x-1

Fig. 7. Placement of an item in bin-packing with enlarged bin capacity.

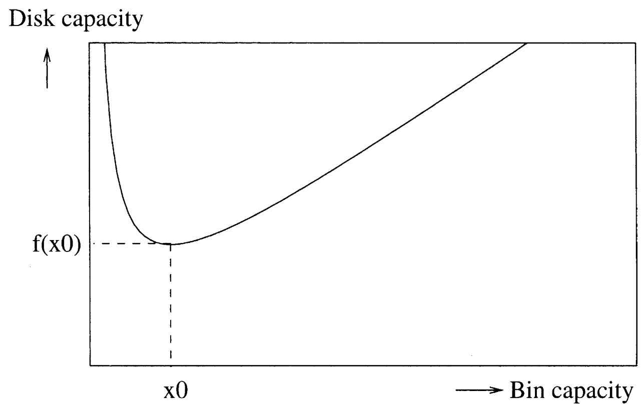


Fig. 8. Curve of the capacity function.

$$f(x_0) = \frac{S}{M} + 2\sqrt{3} * \sqrt{\frac{S}{M}} + 3 \qquad (26)$$

This completes the approximate algorithm.

Theorem 3. [Required disk capacity] *By selecting bin capacity* $x = 1 + \sqrt{S/(3 * M)}$, *the proposed algorithm generates a placement in which*

$$size(D_k) \leqslant \frac{S}{M} + 2\sqrt{3} * \sqrt{\frac{S}{M}} + 3 \qquad (27)$$

for each disk D_k.

Proof. This is the conclusion of previous discussion. □

Theorem 3 relates the capacity required for each disk to the balanced data size S/M, which is a lower bound on the cost of the optimal solution. Eq. (27) thus indicates the relation between the optimal solution and the solution found by the proposed algorithm.

7. LSB_Placement and its asymptotic behavior

We complete the approximate algorithm, *LSB_Placement* (load and storage balanced placement), in Fig. 9. This algorithm determines the bin capacity for the first phase according to the discussion in Section 6. Whether to enlarge the bin capacity or not is determined by comparing the worst case storage requirement stated in Theorem 2 and Theorem 3. Bin capacity is enlarged when Eq. (27) < Eq. (18). The two equations Eq. (18) and Eq. (27) equal at $S/M = 12$ (where S is the total data size and M is the number of disks). Properties of the output are proved in previous sections. Note that, when bin capacity is enlarged (when $S/M > 12$), the selected bin capacity is at least 3 and generated bins are at least 2/3 full except for the smallest bin. The criteria $S/M > 12$ ensures that the derived cost upper bound (Eq. (27)) is tight.

The complexity of the proposed algorithm is as follows. Let N be the number of data items and M be the number of disks. To implement the proposed algorithm, an O($N * M$) space is required to store the intermediate

```
Algorithm $LSB\_Placement(I, D, X)$
  • Input:
      – a set of data items $I = \{I_0, I_1, ..., I_{N-1}\}$ with total data size $S$. Each $I_i$ is
        associated with a size $s_i$ ($0 < s_i \leq 1$) and an access probability $p_i$.
      – a set of disks $D = \{D_0, D_1, ..., D_{M-1}\}$
  • Output: a placement $X$ of items in $I$ onto disks in $D$ with the following properties:
      – $X$ is a load balanced placement.
      – Storage space:
          * if $S/M \leq 12$, $size_X(D_k) \leq 2 * (S/M) + 3$ for any disk $D_k$
          * if $S/M > 12$, $size_X(D_k) \leq (S/M) + 2\sqrt{3} * \sqrt{(S/M)} + 3$ for any disk $D_k$
  • Method:
      1) /* bin-packing phase */
          1.1) if $S/M < 12$ then $x \leftarrow 1$ else $x \leftarrow 1 + \sqrt{S/(3 * M)}$
          1.2) perform bin packing to pack items in $I$ into a set of bins $B$ with bin
               capacity $x$.
      2) /* MMPacking phase */
          2.1) for each bin $B_j \in B$ do $Load(B_j) \leftarrow \sum_{I_i \in B_j} p_i * s_i$
          2.2) perform MMPacking to obtain a placement $Y$ which places bins in $B$
               onto disks in $D$.
      3) for each item $I_i \in I$ do /* obtain the final answer */
          3.1) let $B_j$ be the bin containing $I_i$.
          3.2) for each disk $D_k \in D$ do $X_{ik} \leftarrow Y_{jk}$
```

Fig. 9. Algorithm to generate load and storage balanced placement.

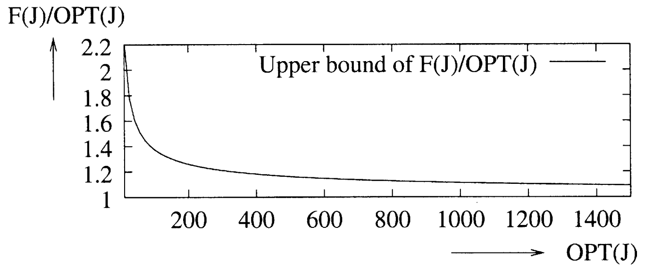


Fig. 10. Ratio to cost of the optimal solution.

placement Y generated by MMPacking and the final placement X. The time complexity of best-fit bin packing is $O(N^2)$ Horowitz et al., 1996. The time complexity for the MMPacking to place $n(\leqslant N)$ bins is $O(n+M)$ Serpanos et al., 1998. Hence the time complexity of the proposed algorithm is $O(N^2+M+n)$.

We now describe the asymptotic behavior of the approximate algorithm. For a problem instance J (a pair of item set I and disk set D), we denote $F(J)$ as the cost (required disk capacity) of the solution found by *LSB_Placement* and $\mathrm{OPT}(J)$ as the cost of the optimal solution. The asymptotic behavior of the optimization problem is as follows:

Theorem 4 (Asymptotic behavior). *LSB_Placement algorithm is* 1*-optimal.*

Proof. We want to show that the ratio $F(J)/\mathrm{OPT}(J)$ approaches 1 as the problem size approaches infinite. When problem size is large (S/M exceeds certain threshold), the cost $F(J)$ is determined by Theorem 3. We have

$$F(J) \leqslant \mathrm{OPT}(J) + 2\sqrt{3} * \sqrt{\mathrm{OPT}(J)} + 3 \tag{28}$$

since $S/M \leqslant \mathrm{OPT}(J)$. And hence

$$1 \leqslant \frac{F(J)}{\mathrm{OPT}(J)} \leqslant 1 + \frac{2\sqrt{3}}{\sqrt{\mathrm{OPT}(J)}} + \frac{3}{\mathrm{OPT}(J)} \tag{29}$$

The fact

$$\lim_{\mathrm{OPT}(J)\to\infty} \left(1 + \frac{2\sqrt{3}}{\sqrt{\mathrm{OPT}(J)}} + \frac{3}{\mathrm{OPT}(J)}\right) = 1 \tag{30}$$

indicates that *LSB_Placement* is 1-optimal. □

The behavior of the proposed algorithm with respect to problem size is depicted in Fig. 10. The upper bound of the curve $F(J)/\mathrm{OPT}(J)$ is indicated by Eq. (29). For a given problem instance J, the point $F(J)/\mathrm{OPT}(J)$ lies between 1.00 and the curve. As the problem size grows (and hence $\mathrm{OPT}(J)$ grows), $F(J)/\mathrm{OPT}(J)$ decreases. The solution found by *LSB_Placement* algorithm approaches optimal when the problem size exceeds certain threshold.

Table 1
Ratio of placement result to optimal storage cost for some typical problem sizes

	$\frac{F(J)}{\mathrm{OPT}(J)}$
$\frac{S}{M} = 12$	<2.25
$\frac{S}{M} = 27$	<1.78
$\frac{S}{M} = 48$	<1.56
$\frac{S}{M} = 75$	<1.45
$\frac{S}{M} = 108$	<1.35
$\frac{S}{M} = 1200$	<1.10

8. Remark on applying to real world applications

This section demonstrates the effectiveness of applying the proposed algorithm to various real world applications. Table 1 shows how the ratio $(F(J)/\mathrm{OPT}(J))$ is decreased according to Eq. (27). Let S be the total data size of all items (with size of the largest item normalized to 1.00) and M be the number of disks. The balanced storage size S/M is chosen to be 1, 1.5^2, 2^2, 2.5^2, 3^2,10^2 times $2\sqrt{3}^2$. These selections of S/M are to compare the storage requirement (Eq. (27)) to $((1+(1/a)) * (S/M) + \text{constant})$, where a is an integer or a simple rational number. A simple way to check whether *LSB_Placement* can generate a load balanced placement for a given configuration is to compute the order of S/M and map it in Table 1.

Table 2 shows the performance of applying *LSB_Placement* to various Web applications (IEEE/IEE [1], ACM [2] and MP3.COM [3]). In late 1990s, the capacity of a commercial disk is 10–50 GB and the size of a single file is small compared to the disk capacity. It is appropriate to use a number of disks such that, in the optimal placement, a disk contains thousands of files and S/M is in the range of several thousands. For each of the listed applications, a load balanced placement can be generated as long as the capacity of each disk is 10% more than the balanced storage space.

9. Conclusion

This paper investigates the optimization problem of placing variable-size data items onto multiple disks with replication. The objective is to minimize storage cost subject to the ideal load balancing constraint. A data placement algorithm is proposed. The algorithm performs bin packing followed by MMPacking to place variable-size items onto workstations. We prove that the algorithm generates a load balanced placement with asymptotically 1-optimal storage cost.

[1] IEEE/IEE electronics library (http://iel.ihs.com).
[2] ACM digital library (http://www.acm.org/dl).
[3] MP3.COM (http://mp3.com).

Table 2
Performance of LSB_Placement algorithm for some Web applications

File type	Typical file size	Range of $\frac{S}{M}$	$F(J)/\mathrm{OPT}(J)$
Papers (pdf or postscript)	300 KB–1.5 MB	>1200	$\leqslant 1.10$
Homepages	10 KB–2 MB	>1200	$\leqslant 1.10$
MP3 music	1 MB–20 MB	>1200	$\leqslant 1.10$

These results greatly simplify the design of a large-scale file server. In recent years, many major Websites use a cluster with huge storage volume to cope with high request arrival rate and store all data. Simulation, the traditional method to design a computer system, is infeasible for such a large-scale file server. A quantitative method that precisely estimate the performance with closed form equations is desired. A barrier to derive a quantitative method was the lack of a data placement algorithm with good ϵ-optimality been proved. This paper presents the first ϵ-optimality result for the concerned data placement problem. The result shows that, even in the worst-case, the difference between generated placement and optimal placement is still small and acceptable. This paper opens the quantitative approach to design a large-scale file server. Papadimitriou and Steiglitz (1982); Coffman et al. (1978); Garey and Johnson (1979) and Kwan et al. (1995).

References

Coffman, E.G., Garey, M.R., Johnson, D.S., 1978. An application of bin-packing to multiprocessor scheduling. SIAM Journal on Computing 7 (1), 1–17.

Dowdy, W., Foster, D., 1982. Comparative models of the file assignment problem. ACM Computing Surveys 14 (2), 287–313.

Garey, M., Johnson, D., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness.

Horowitz, E., Sahni, S., Rajasekaran, B., 1996. Computer Algorithms/ C++.

Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R., Graham, R.L., 1974. Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM Journal on Computing 3 (4), 299–325.

Kwan, T., Mcgrath, R., Reed, D., 1995. NCSA world wide web. Computer 28 (11), 67–74.

Lee, H., Park, T., 1995. Allocating data and workload among multiple servers in a local area network. Information Systems 20 (3).

Lee, L.W., Scheuermann, P., Vingralek, R., 2000. File assignment in parallel i/o systems with minimal variance of service time. IEEE Transactions on Computers 49 (2), 127–140.

Little, T.D.C., Venkatesh, D., 1995. Popularity-based assignment of movies to storage devices and video-on-demand system. ACM/ Springer Multimedia System 2 (6), 280–287.

Narendran, B., Rangarajan, S., Yajnik, S., 1997. Data distribution algorithm for load balanced fault-tolerant web access. In: Proceedings of 16th Symposium on Reliable Distributed Systems. pp. 97–106.

Papadimitriou, C.H., Steiglitz, K., 1982. Combinatorial Optimization: Algorithms and Complexity.

Rotem, D., Schloss, G., Segev, A., 1993. Data allocation of multidisk databases. IEEE Transactions Knowledge and Data Engineering 5 (5), 877–882.

Serpanos, D.N., Georgiadis, L., Bouloutas, T., 1998. MMPacking: a load and storage balancing algorithm for distributed multimedia servers. IEEE Transactions on Circuits and Systems for Video Technology 8 (1), 13–17.

Wah, B., 1984. File placement on distributed computer systems. Computer 17 (1), 23–32.

Wolf, J., Pattipati, K., 1990. A file assignment problem model for extended local area network environments. Proceedings of 10th International Conference on Distributed Computing Systems 17 (1).

Yung-Cheng Ma received the B.S. degree in computer science and information engineering from the National Chiao-Tung University, Hsinchu, Taiwan, Republic of China in 1994. Currently he is pursuing the Ph.D. degree in computer science and information engineering at the National Chiao-Tung University, Hsinchu, Taiwan, Republic of China. His research interests include computer architecture, parallel and distributed systems, optimization algorithms, and Web information systems.

Jih-Ching Chiu received the B.S. degree in Electrical Engineering from National Sun Yet-Sen University in 1984, and M.S. degree in Electrical Engineering from National Cheng Kung University in 1986. He is now working towards the Ph.D. degree in Department of Computer Science and Information Engineering of National Chiao Tung University at Hsinchu from 1994. His research interests include processor architectures and microprocessor-based systems.

Tien-Fu Chen received the B.S. degree in Computer Science from National Taiwan University in 1983. After completed his military services, he joined Wang Computer Ltd., Taiwan as a software engineer for three years. From 1988 to 1993 he attended the University of Washington, receiving the M.S. degree and Ph.D. degrees in Computer Science and Engineering in 1991 and 1993 respectively. He is currently an Associate Professor in the Department of Computer Science and Information Engineering at the National Chung Cheng University, Chiayi, Taiwan, In recent years, he has published several widely cited papers on dynamic hardware prefetching algorithms and designs. His current research interests are computer architectures, distributed operating systems, parallel and distributed systems, and performance evaluation.

Chung-Ping Chung received the B.E. degree from the National Cheng-Kung University, Hsinchu, Taiwan, Republic of China in 1976, and the M.E. and Ph.D. degrees from the Texas A&M University in 1981 and 1986, respectively, all in electrical engineering. He was a lecturer in electrical engineering at the Texas A&M University while working towards the Ph.D. degree. Since 1986 he has been with the Department of Computer Science and Information Engineering at the National Chiao-Tung University, Hsinchu, Taiwan, Republic of China, where he is a professor. From 1991 to 1992, he was a visiting associate professor of computer science at the Michigan State University. From 1998, he joint the Computer and Communications Laboratories, Industrial Technology Research Institute, R.O.C. as the Director of the Advanced Technology Center, and then the Consultant to the General Director. He is expected to return to his teaching position after this three-year assignment. His research interests include computer architecture, parallel processing, and parallelizing compiler.