COMPUTER STUDIES PUBLICATION No TR-A3-84

Technical Report

MULTIDISK FILE DESIGN :

AN ANALYSIS OF FOLDING BUCKETS TO DISKS

M Y CHAN

HKU CCSA

CENTRE OF COMPUTER STUDIES AND APPLICATIONS

UNIVERSITY OF HONG KONG

POKFULAM ROAD

HONG KONG

FOREWORD

The present Technical Report is the preliminary version of a paper which is currently under consideration for publication in a journal. Prior to acceptance and publication of the paper, its copyright is vested in the author. As a courtesy to its prospective publisher, copying of the Report should be kept to the minimum essential requirement.

The ideas expressed in the Report represent those of the author, and should not be taken as the official views of the Centre or the University. Discussion of its contents and request for additional copies should be directed to the author at the address shown on the cover.

C K Yuen

Director of Computer Studies

Multidisk File Design : An Analysis of Folding Buckets to Disks

by

Mee Yee Chan

Centre of Computer Studies and Applications

University of Hong Kong

Abstract

A technique called folding for mapping file buckets to multiple disks is evaluated. In particular, an upper bound for expected costs given any size partial match query is found. Folding is compared against Disk Modulo allocation.

Introduction

This paper relates to partial match retrieval for large, on-line data files spread across several independently accessible disks, a concern first introduced by Du and Sobolewski [1] as having relevance for database information retrieval. The problem in question is how should such files be arranged among the disks to best facilitate queries, exploiting to the fullest the concurrency of access on separate disks. We introduce an arrangement technique called folding to tackle multidisk file design under a binary framework, making rough comparisons with Disk Modulo allocation [1] in an attempt to show folding as viable.

The problem we wish to consider in this paper can be described as follows. We are interested in three entities : buckets, disks, and partial match queries. A bucket is a package of information, each bucket keyed uniquely by a d-dimensional vector of 0's and 1's. A partial match query is a request to retrieve a set of buckets, each query also denoted by a d-dimensional vector but of 0's, 1's, and *'s. Buckets which satisfy a query agree with the query vector in positions having 0 or 1 with *'s representing "don't care" positions. A disk is a location to which we can assign buckets, each disk addressed by a m-dimensional vector of 0's and 1's. A multidisk file design is a mapping of the 2^d buckets to the 2^m disks. The cost of a partial match query given a multidisk design is the maximum of the number of buckets, which satisfy the query, for each disk. The optimal (minimal) cost for a query with p *'s is $\lceil 2^{p-m} \rceil$. The multidisk file design problem involves the construction of a design which minimizes cost on average over all possible partial match queries.

As an example of the various terms just introduced, consider the multidisk file designs shown in Figures 1 and 2 for d=6, m=3. Under the design of Figure 1, queries ******, *0*1*0, and *0**11 yield costs of 20, 3, and 3, respectively, whereas applying these queries to Figure 2 yields respective costs of 8, 1, and 2. The query ****** is essentially a request for all buckets. To obtain all buckets in the Figure 1 design, 1,6,15,20,15,6,1,0 buckets need be respectively gotten from Disks 1 through 8; hence, a cost of 20 is incurred. In the Figure 2 situation, 8 buckets per disk determine a cost of 8. Figures 3 and 4 explain the expense associated with *0*1*0 and *0**11. Note the optimality of the Figure 2 assignment for ****** and *0*1*0.

Since any bucket key can be encoded into binary form and binary partial match queries allow the user greatest flexibility in specifying queries, our binary framework is noteworthy. Unsurprisingly, Disk Modulo allocation performs its worst for this problem.

<u>An Analysis of Disk Modulo</u>

The problem of assigning buckets to disks to achieve the best average performance is a difficult one to solve in general. Past research has given us a heuristic in the form of Disk Modulo allocation. Disk Modulo when applied to our multidisk problem effectively maps buckets according to however many 1's appear in the bucket vector; a bucket with i 1's finds itself assigned to the (i mod 2^m + 1)th disk. Figure 1 is, in fact, an instance of the Disk Modulo technique. Rather awkward mappings are seen to result where certain disks hold many more buckets than others. Indeed, the following may be asserted.

Lemma. When a Disk Modulo multidisk file design is used, any partial match query with p *'s will cost at least $\binom{p}{\lceil p/2 \rceil}$ buckets.

Proof of Lemma. Given a p-* query, let q be the number of 1's appearing in the query vector. To arrive at the buckets satisfying such a query is essentially an exercise in replacing the *'s with either 0 or 1. For example, query *0*1*0 asks for bucket 000100 gotten from replacing all 3 *'s by 0, buckets 000110, 001100, 100100 gotten from replacing exactly 2 *'s by 0, buckets 001110, 100110, 101100 gotten from replacing exactly 1 * by 0, and bucket 101110 gotten by substitution with no 0's (all 1's). In general, there are $\binom{p}{i}$ buckets obtained by substituting exactly (p-i) *'s with 0, all of which are located on the $((q+i) \bmod 2^m + 1)$th disk. Hence, for a m=3 disk environment, buckets 000110, 001100, 100100 are all found on Disk 3. Since $\max_i \binom{p}{i} = \binom{p}{\lceil p/2 \rceil}$, the maximum of the number of relevant buckets for each disk is at least $\binom{p}{\lceil p/2 \rceil}$. The "at least" takes into consideration cases where $((q+i) \bmod 2^m + 1) = ((q+j) \bmod 2^m + 1)$ for some integers i,j, $i \neq j$ and $0 \leq i,j \leq p$. □

An Analysis of Folding

The main idea of this paper is to investigate a method, which we call "folding", as an alternative to Disk Modulo allocation. The design of Figure 2 illustrates how buckets are folded to disks. The technique itself is actually borrowed from the traditional hashing concept of folding : in mapping d-bit keys to m-bit addresses, assuming d = km for some integer k, the d-bit key is partitioned into k m-bit parts which are added together,ignoring any final carry, to obtain the necessary m-bit address [3]. Hence, the steps for folding 6-bit bucket 100110 to a 3-bit address include adding 100 to 110 (in the binary sense) to obtain 1010 and ignoring carry to yield 3-bit address 010 (Disk 3). Likewise, a 9-bit bucket 010011101 invokes the addition of 010, 011, 101 with a disregard of the resultant carry to get a 3-bit address of 000 (Disk 1).

The main result of our analysis is given by the following theorem.

Theorem. Let A(p) denote the expected cost over all queries with p *'s (equally probable) given a folded, m-bit disk address, d-bit bucket key (d = km for some integer k) system. Then,

$$A(p) \leq \left\{ \sum_{i=0}^{p} \binom{m}{i} 2^{p-i} \sum_{j=0}^{i} \binom{i}{j} \binom{(i-j)d/m}{p} (-1)^j \right\} \Big/ \binom{d}{p} .$$

Proof of theorem. In proof we begin with the notion of cost classes. A cost class is a m-bit vector that represents a set of queries. A query

$$q_{11}q_{12}\cdots q_{1m}q_{21}q_{22}\cdots q_{2m}\cdots q_{k1}q_{k2}\cdots q_{km}$$

belongs to cost class $a_1a_2\cdots a_m$

if and only if

$$a_t = \begin{cases} 1 & \text{if } q_{st} = * \text{ for some } s \\ 0 & \text{otherwise} \end{cases} , \text{ for } t = 1,2,\ldots,m.$$

We prove two lemmas in conjunction with cost classes, the first bounds the cost of p-* queries within a cost class and the second enumerates the number of p-* queries within a class.

Lemma 1. In a folded multidisk design, any p-* query in cost class $a_1a_2\cdots a_m$ costs at most 2^{p-i} where $i = \sum_{t=1}^{m} a_t$.

Proof of Lemma 1. The argument is that any p-* query in cost class $a_1 a_2 \ldots a_m$ can be answered by posing 2^{p-i} different i-* queries, all of which have *'s in the same positions and belong to $a_1 a_2 \ldots a_m$, for i as defined by the lemma. To respond to *0**11 for a d=6,m=3 folded design, we can pose *0*011 and *0*111 both of cost class 101. Further, any such i-* query can be satisfied using a cost of 1. Thus, the cost expected for the p-* query is at most 2^{p-i}. Since one of the i-* requests may require a particular disk be accessed while another may not, we may fare better than 2^{p-i}, hence the "at most". □

Lemma 2. The number of p-* queries in cost class $a_1 a_2 \ldots a_m$ is

$$2^{d-p} \sum_{j=0}^{i} \binom{i}{j} \binom{(i-j)d/m}{p} (-1)^j$$

where $i = \sum_{t=1}^{m} a_t$.

Proof of Lemma 2. We apply the principle of inclusion and exclusion in proof [2] . Let $\{r_1, r_2, \ldots, r_i\}$ be the set of indices of cost class $a_1 a_2 \ldots a_m$ for which $a_{r_t} = 1$ for $t = 1,2,\ldots,i$. Consider the set S of p-* queries such that if q_{sr}=* then $r \in \{r_1, r_2, \ldots, r_i\}$. There are $N = 2^{d-p} \binom{id/m}{p}$ queries in S. Let A_{r_t} be the property that a query has q_{sr_t}=* for no s and let A'_{r_t} be the property that a query has q_{sr_t}=* for some s. The notation N(X) denotes the number of queries in S having the list of properties X. So $N(A'_{r_1} A'_{r_2} \ldots A'_{r_i})$ denotes the number of queries in S with properties $A'_{r_1}, A'_{r_2}, \ldots, A'_{r_i}$ and is in fact the number of p-* queries in cost class $a_1 a_2 \ldots a_m$.

$$N(A'_{r_1} A'_{r_2} \cdots A'_{r_i}) = N - \sum_{t=1}^{i} N(A_{r_t}) + \cdots + (-1)^i N(A_{r_1} A_{r_2} \cdots A_{r_i})$$
$$= 2^{d-p}\left[\binom{i}{0}\binom{id/m}{p} - \binom{i}{1}\binom{(i-1)d/m}{p} + \cdots + (-1)^i \binom{i}{i}\binom{0}{p}\right]. \square$$

There are $2^{d-p}\binom{d}{p}$ p-* queries altogether. The upper bound for expected costs of the theorem follows naturally from these two lemmas.

Folding vs. Disk Modulo

Now that we have a bound on the expected cost for any size query given a folding-based multidisk design, how viable is folding? We conclude with the tabulations of Figures 5 and 6. When compared with the lower bound of $\binom{p}{\lceil p/2 \rceil}$ for Disk Modulo, the upper bound from the theorem is seen to be often less and closer to optimal costs of $\lceil 2^{p-m} \rceil$. These numerical comparisons, unfortunately in the absence of definite relative cost statements, suffice to render folding as a worthwhile approach to bucket-to-disk file design.

References

1 H. C. Du and J. S. Sobolewski. Disk allocation for Cartesian product files on multiple-disk systems. ACM Trans. Database Syst. 7, 1 (March 1982), pp. 82-101.

2 C. L. Liu, Introduction to Combinatorial Mathematics, McGraw-Hill, New York, 1968.

3 J. P. Tremblay and P. G. Sorenson, An Introduction to Data Structures with Applications, McGraw-Hill, New York, 1976.

Disk :	1	2	3	4	5	6	7	8
	000	001	010	011	100	101	110	111
Buckets	000000	000001	000011	000111	001111	011111	111111	
		000010	000101	001011	010111	101111		
		000100	000110	001101	011011	110111		
		001000	001001	001110	011101	111011		
		010000	001010	010011	011110	111101		
		100000	001100	010101	100111	111110		
			010001	010110	101011			
			010010	011001	101101			
			010100	011010	101110			
			011000	011100	110011			
			100001	100011	110101			
			100010	100101	110110			
			100100	100110	111001			
			101000	101001	111010			
			110000	101010	111100			
				101100				
				110001				
				110010				
				110100				
				111100				

Figure 1. Disk Modulo allocation of 6-bit buckets to 3-bit disks

Disk :	1	2	3	4	5	6	7	8
	000	001	010	011	100	101	110	111
Buckets	000000	000001	000010	000011	000100	000101	000110	000111
	001111	001000	001001	001010	001011	001100	001101	001110
	010110	010111	010000	010001	010010	010011	010100	010101
	011101	011110	011111	011000	011001	011010	011011	011100
	100100	100101	100110	100111	100000	100001	100010	100011
	101011	101100	101101	101110	101111	101000	101001	101010
	110010	110011	110100	110101	110110	110111	110000	110001
	111001	111010	111011	111100	111101	111110	111111	111000

Figure 2. Folding allocation of 6-bit buckets to 3-bit disks

Disk :	1	2	3	4	5	6	7	8
	000	001	010	011	100	101	110	111
		000100	000110	001110	101110			
			001100	100110				
			100100	101100				

(a)

Disk :	1	2	3	4	5	6	7	8
	000	001	010	011	100	101	110	111
			000011	000111	001111	101111		
				001011	100111			
				100011	101011			

(b)

Figure 3. Relevant buckets for (a) *0*1*0 and (b) *0**11 from the multidisk design of Figure 1.

Disk :	1	2	3	4	5	6	7	8
	000	001	010	011	100	101	110	111
	100100	101100	100110	101110	000100	001100	000110	001110

(a)

Disk :	1	2	3	4	5	6	7	8
	000	001	010	011	100	101	110	111
	001111			000011	001011			000111
	101011			100111	101111			100011

(b)

Figure 4. Relevant buckets for (a) *0*1*0 and (b) *0**11 from the multidisk design of Figure 2.

m	d	p	(i)	(ii)	(iii)
2	4	1	1.00	1	1
2	4	2	1.33	2	1
2	4	3	2.00	3	2
2	4	4	4.00	6	4
2	6	1	1.00	1	1
2	6	2	1.40	2	1
2	6	3	2.20	3	2
2	6	4	4.00	6	4
2	6	5	8.00	10	8
2	6	6	16.00	20	16
2	8	1	1.00	1	1
2	8	2	1.43	2	1
2	8	3	2.29	3	2
2	8	4	4.11	6	4
2	8	5	8.00	10	8
2	8	6	16.00	20	16
2	8	7	32.00	35	32
2	8	8	64.00	70	64
2	10	1	1.00	1	1
2	10	2	1.44	2	1
2	10	3	2.33	3	2
2	10	4	4.19	6	4
2	10	5	8.06	10	8
2	10	6	16.00	20	16
2	10	7	32.00	35	32
2	10	8	64.00	70	64
2	10	9	128.00	126	128
2	10	10	256.00	252	256
2	12	1	1.00	1	1
2	12	2	1.45	2	1
2	12	3	2.36	3	2
2	12	4	4.24	6	4
2	12	5	8.12	10	8
2	12	6	16.03	20	16
2	12	7	32.00	35	32
2	12	8	64.00	70	64
2	12	9	128.00	126	128
2	12	10	256.00	252	256
2	12	11	512.00	462	512
2	12	12	1024.00	924	1024

Figure 5. Folding vs. Disk Modulo :
(i) upper bound for expected cost for folding
(ii) lower bound for expected cost for Disk Modulo
(iii)optimal cost
as described in the paper.

m	d	p	(i)	(ii)	(iii)
3	6	1	1.00	1	1
3	6	2	1.20	2	1
3	6	3	1.60	3	1
3	6	4	2.40	6	2
3	6	5	4.00	10	4
3	6	6	8.00	20	8
3	9	1	1.00	1	1
3	9	2	1.25	2	1
3	9	3	1.75	3	1
3	9	4	2.71	6	2
3	9	5	4.57	10	4
3	9	6	8.29	20	8
3	9	7	16.00	35	16
3	9	8	32.00	70	32
3	9	9	64.00	126	64
3	12	1	1.00	1	1
3	12	2	1.27	2	1
3	12	3	1.82	3	1
3	12	4	2.86	6	2
3	12	5	4.85	10	4
3	12	6	8.73	20	8
3	12	7	16.48	35	16
3	12	8	32.19	70	32
3	12	9	64.00	126	64
3	12	10	128.00	252	128
3	12	11	256.00	462	256
2	12	12	512.00	924	512
3	15	1	1.00	1	1
3	15	2	1.29	2	1
3	15	3	1.86	3	1
3	15	4	2.95	6	2
3	15	5	5.01	10	4
3	15	6	9.01	20	8
3	15	7	16.90	35	16
3	15	8	32.67	70	32
3	15	9	64.38	126	64
3	15	10	128.13	252	128
3	15	11	256.00	462	256
3	15	12	512.00	924	512
3	15	13	1024.00	1716	1024
3	15	14	2048.00	3432	2048
3	15	15	4096.00	6435	4096

Figure 6. Folding vs. Disk Modulo :
(i) upper bound for expected cost for folding
(ii) lower bound for expected cost for Disk Modulo
(iii)optimal cost
as described in the paper.

001.6442

Chan, M.Y.

Multidisk file design. 1984.