This is to certify that the

dissertation entitled

DATA DISTRIBUTION STRATEGIES
FOR PARALLEL DATABASE ACCESSES

presented by

Myoungho Kim

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Computer Science

B. Ramamurthy
Major professor

Date 2/2/89

MSU is an Affirmative Action/Equal Opportunity Institution O-12771

DATA DISTRIBUTION STRATEGIES FOR PARALLEL DATABASE ACCESSES

By

Myoungho Kim

A DISSERTATION

Submitted to

Michigan State University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1989

ABSTRACT

DATA DISTRIBUTION STRATEGIES FOR PARALLEL DATABASE ACCESSES

By

Myoungho Kim

With the advent of commercially available general purpose multiprocessing systems, the need for developing appropriate information processing systems are increasingly recognized. Since many database applications require a large number of data accesses with relatively less computation, exploiting parallel data accesses is important to improve performance in parallel processing database systems. In this thesis we investigate data distribution strategies for parallel processing of database systems. The primary objective is to maximize throughput and minimize response time through concurrent data accesses. We propose database processing models as a general framework, and then present data distribution strategies for three common types of database applications. Two of these applications are on multikey hash files and the third application is on B-tree accesses. First, we present data distribution strategies for partial match queries. The main contribution here is the development of new data distribution methods called Fieldwise eXclusive-or (FX) distribution methods to achieve optimal file distribution. An algebraic property of exclusive-or operation along with field transformation techniques are fundamental to these data distribution methods. We show that the proposed data distribution methods perform better than the others proposed in the past. We also present efficient data construction methods based on the usage of multikey hash directory. Second, optimal distribution for parallel processing of multiattribute range queries is investigated. Here, we show that for various types of

multiattribute range queries there are inherent limitations in achieving optimal distribution. We extend FX distribution methods to achieve optimal distribution for many useful multiattribute range queries. For both partial match queries and multiattribute range queries, sufficient conditions for optimal distribution by the proposed distribution methods are given. Finally, we present node partitioning schemes for B-tree type indexes. The objective here is to develop a new parallel processing scheme for B-tree type indexes stored in parallel disks. We show that parallel processing of the proposed partitioned node B-trees performs better than parallel processing of conventional B-trees. This work presents a new basis on which parallel processing systems for other database applications can be designed.

To my parents Han Cheol Kim and

Kye Soon Lee

ACKNOWLEDGEMENTS

I wish to express my appreciation to my advisor, Dr. Sakti Pramanik, for his guidance over the years. Dr. Pramanik provided valuable comments and constant support in every phase of the preparation of this work. I would also like to thank Dr. Lionel M. Ni and Dr. Abdol Esfahanian for their many valuable suggestions and comments. I also wish to express my appreciation to Dr. Dorian Feldman for his encouragement and support. Without their intellectual advice and inspiration, this dissertation would not have been possible. I would like to thank all the people who helped me during my stay at Michigan State University.

Finally, I would like to thank my parents, my wife and my son for their constant support, patience, and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

There will be many applications for large databases that cannot be performed in an acceptable response time by current database systems. Since one of the most significant ways of improving performance is through parallelism, the importance of parallel processing in database systems has been increasingly recognized. Parallel processing in database systems can increase throughput and minimize response time through concurrent data accesses as well as processing data in parallel. However, parallel processing by itself does not necessarily lead to high performance. Some of the reasons are attributed to overhead due to interprocessor communication, remote memory accesses and data access conflicts. For parallel database operations external I/O also causes a serious performance bottleneck [Bor83]. Stone [Sto87] shows that parallel query algorithms in a multiprocessor system may perform poorly than efficient serial algorithms on a single processor system. The advantage of indexing to reduce external I/O traffic was also emphasized in that paper. There have been numerous works on improving performance of database systems by specialized software/hardware techniques. These include database machines, main memory resident databases and concurrent dynamic search structures.

1.1. Database Machines

Many proposals have been made in the past for machine architectures for efficient parallel processing of database operations. Early designs such as CASSM [Su79] and RAP [Ozk75] exploit the logic per track idea which was first proposed in [Slo70] as an alternative to pure associative memories. They are SIMD architectures and focus on

parallel scanning of data on secondary storage devices. DIRECT [Dew79] is a logical extension of the SIMD type associative processors which supports both intra-query and inter-query concurrency. It assigns the number of processors dynamically to a query and so it allows users to share the processing power simultaneously. These machines can be characterized by functional replication approach because processors (or memories) are basically similar to each other and capable of performing the same set of functions in parallel. On the other hand, database machines such as RDBM [Scvh83] and SiDBM [Lel85] have used functional specialization approach which means that different functional units are used for different classes of database operations.

Some database machines such as DBMAC [Mis83] and Delta [Kak85] use an attribute based data model which employs vertical fragmentation concept. They use a vertical fragmentation of a relation to minimize the cost of processing and transferring unnecessary attributes from disks. Other database machines which use some form of vertical fragmentation are DBC [Ban79] and a database computer proposed in [Tan83].

Due to the similarity between relational algebra tree and data flow graph, the data driven computation concept has been proposed as an effective operational method for relational database machines [Bor82]. GAMMA, proposed in [Dew86], exploits dataflow query processing techniques. In GAMMA, queries are compiled into a tree of operators, and each operator is executed by one or more operator processes. With the exception of a few control messages at the time a query is initiated, execution of an operator is self-scheduling, i.e., when the process terminates execution, data flows between the processes without centralized control. There are also other proposals of database machines using data flow concepts [Bic83, Gli83].

Performance of various database machines are discussed in [Haw82, Hil86]. The results show that the performance improvement depends on the query type as well as the database machine architecture.

1.2. Main Memory Databases

Since inexpensive large main memory is becoming available, and many database applications are I/O bound, some work has been done on main memory database systems to minimize data access time. ESP, presented in [GarL84], consists of multiple processors, but it operates in SISD mode. ESP uses single data stream with all processors operating in lock step. The main focus of this machine is to reduce the global memory access time. SiDBM presented in [Lel85], on the other hand, has an MIMD architecture in which the processors are functionally specialized. Both machines use shared memory with common bus architecture.

Storage structure for main memory databases has been studied in the past. Dewitt et al. discussed the access time of AVL trees and B-trees for various memory residency factors of databases [Dew84]. They have also shown that hash based join algorithms are very effective for large main memory database systems. Lehman et al. give performance results of main memory databases using various types of access methods [Leh86]. In that paper they have also proposed a T tree as an alternative to other tree structures. The T tree is a balanced binary tree with many elements per node. It exploits binary search nature of AVL trees, and good update and storage characteristics of B-trees. Several other issues of main memory databases have also been investigated in [Amm85].

1.3. Concurrent Dynamic Search Structures

There have been several proposals on concurrent dynamic search structures to support parallel processing. Many of them are based on tree structures such as B-trees and binary search trees. For concurrent B-tree operations, the entire subtree of the highest affected node is exclusively locked in [Sam76]. Bayer et al. present the algorithms which only lock out other writers until the actual modifications must be performed [Bay77]. This approach distributes exclusive locks mostly in lower sections of the B-

tree and hence it increases the concurrency. Lehman et al. also described algorithms for concurrent B-tree operations which use only a small number of locks for each tree manipulating process [Leh81]. Concurrent search and insertion algorithms for AVL trees and 2-3 trees are given in [Ell80-a, Ell80-b]. There are also several other work on concurrent manipulation of binary search trees [Kun80, Man84].

1.4 Classification of Database Queries

Parallel processing strategy which is appropriate for one database application may not be appropriate for another. For example, the granularity of parallel processing which is suitable for one application may not be so for others. Thus, it may be necessary to develop different parallel processing strategies for various types of applications. We classify database queries based on their parallel processing characteristics, as follows :

(A1) Single query with multiple hits

(A2) Single query with a single hit

(A3) Single complex query

(A4) Multiple queries accessing the same relation

(A5) Multiple queries accessing different relations

Examples of queries of type A1 are partial match queries and range queries. Here, intra-query parallel processing is advantageous because a single query requiring many data records can be processed in parallel [Kim88, Pra88-b, Pra88-d]. Rosenau et al. have also applied this type of parallel processing for projection operation on a relation [Ros87].

It is rather difficult to exploit parallel processing of type A2 queries because applying parallelism for this type of query may require finer granularity which may result in lower throughput of the system. On the other hand, parallel processing may achieve the

lower bound on access time for these types of queries when appropriate software and hardware architecture is used. Achieving and guaranteeing this lower bound are important for many real-time critical applications. Parallel processing strategies for this type of applications can be found in [Pra86, Pra87, Pra88-a, Pra88-c].

Queries of type A3 include join functions, sorting of files, and complex qualifications. Several database machines which use functionally distributed architectures have been proposed for this type of queries [Lel85, Sch83]. Intra-query parallelism is also advantageous for this type of applications.

For A4 and A5 type queries, many independent queries can be processed in parallel. Transaction processing is an example for these types. The throughput of the system for these types of applications can be improved by maximizing concurrency among the queries. Multidirectory hashing proposed in [Cev88] has been shown to be effective for certain applications of these types.

1.5. Problem Statement

The primary objective of this work is to investigate data distribution strategies for various types of database applications in parallel processing systems. In this thesis we will mainly focus on the data distribution strategies for A1 and A2 type queries because these queries are in fact the most basic types, and data distribution strategies for other types of applications may be developed based on the strategies used in A1 and A2 type queries. For queries of type A1, we will investigate optimal data distribution for multikey search queries such as partial match retrieval and multiattribute range queries. Since multikey search queries access a set of records, appropriate data distribution is very important to facilitate parallel data accesses. Though much work has been done in the past on designing efficient file structures for this type of applications, data distribution to enhance concurrency has not been considered much. We will investigate optimal data distribution and appropriate data construction for this type of applications.

For type A2 queries, we will investigate parallel processing of tree type indexes for external databases. Since the B-tree is a common storage structure for this type of applications, node partitioning schemes to exploit parallel data accesses to B-trees will be investigated.

1.5. Thesis Overview

The remainder of this thesis is organized as follows. In chapter 2 we describe high level abstraction of database processing for parallel processing systems. The objective of this abstraction is to define a framework which can be used as a basis for more specific implementation.

In chapter 3 we present optimal data distribution for partial match retrieval type queries. The main focus of this chapter is the development of new data distribution methods, called Fieldwise eXclusive-or (FX) distribution methods, for maximizing data access concurrency. FX distribution methods use bitwise exclusive-or operation on the field values which are computed by multikey hashing. We show several useful characteristics of exclusive-or operation for optimal file distribution. Field transformation techniques are presented to extend the scope of optimality in FX distribution methods. We give sufficient conditions for optimal distribution by the proposed distribution methods and show that these methods are optimal for most partial match queries. We describe the performance improvement by FX distribution methods over other methods proposed earlier. Efficient data construction methods for FX distribution approach are also discussed.

In chapter 4 we investigate file distribution problems for parallel processing of multiattribute range queries. Optimal data distribution methods as well as the existence and nonexistence of perfect optimality are investigated. It will be shown that there are inherent limitations to achieve optimal distribution for various types of range queries. We give sufficient conditions for the nonexistence of perfect optimal distribution for

certain types of range queries. We extend the FX distribution methods for several useful multiattribute range queries. Sufficient conditions for optimal distribution for multiattribute range queries will be described. It will be shown that the proposed data distribution methods are optimal for a large class of multiattribute range queries.

In chapter 5 we investigate the performance of various parallel processing methods for B-trees. The main focus of this chapter is a node partitioning scheme for B-trees to exploit parallel data accesses. The proposed B-tree node partitioning scheme is based on synchronized disks. Parallel processing of partitioned node B-trees on asynchronous disks are also discussed. We also show the performance improvement of partitioned node B-trees over conventional B-trees. Chapter 6 contains concluding remarks.

CHAPTER 2

DATABASE PROCESSING MODELS FOR PARALLEL PROCESSING

In this chapter we describe high level abstraction of database processing for parallel processing systems. The objective of this abstraction is to define a framework which can be the basis of more specific implementation.

2.1. General Parallel Processing Model for Database Systems

We propose an abstract model for parallel processing of database systems. This is shown in Figure 2.1. The model is based on distributing data and access structures to enhance concurrency. In the figure, Q_i's represent a set of parallel access nodes. These can be main memory modules or disks, depending on the parallel processing environment. A_i's denote a set of access structures. As shown in the figure, there are three mappings that are important for concurrent processing. They are (1) storage allocation for data (2) storage allocation for access structures and (3) key to access structure mapping.

Storage allocation for data, and storage allocation for access structures determine the amount of physical access concurrency among the nodes. In practice, these issues should be considered based on the architectural types of parallel processing systems. In other words, a number of factors such as the interconnection structure between the processors and the access nodes may contribute to actual system performance. In this thesis we assume that the parallel access nodes are logically single shared memory.

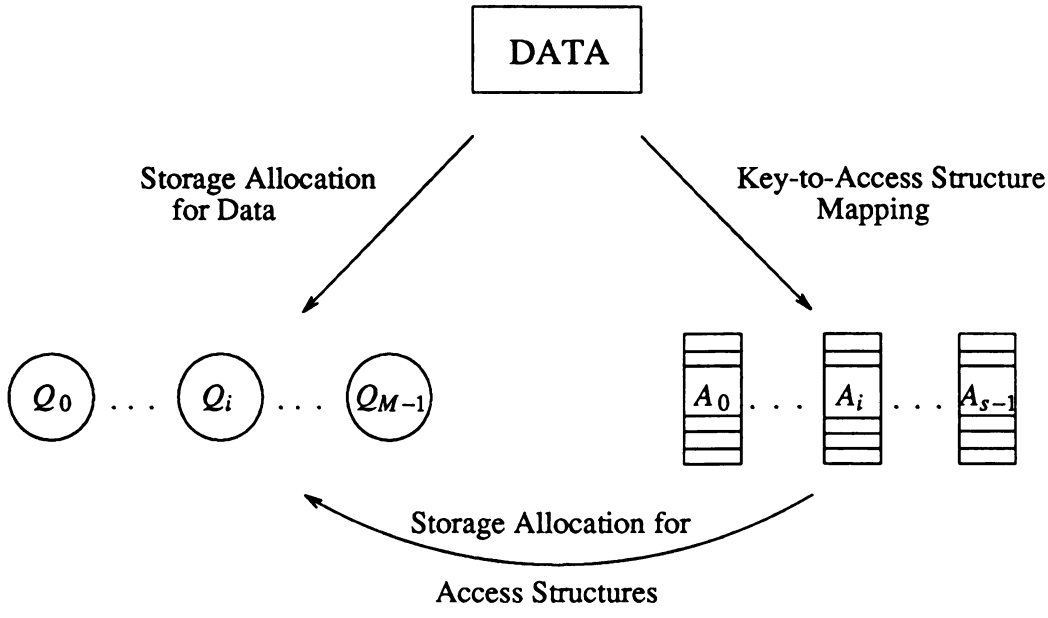


Figure 2.1. General Parallel Processing Model

In the parallel processing system, the speed mismatch between computation and data access time becomes a more serious problem than in the uniprocessor system due to data access conflict if data are not evenly distributed over access nodes. Generally, database applications require many data accesses with relatively less computation, and so degradation of system performanc due to inappropriate storage allocation scheme may be more significant in database applications than in others.

In addition to physical memory (or device) access conflict, there are other sources of conflict for concurrent accessing of data. This means that even though two data are stored in different access nodes, access concurrency can still be reduced. One such source is the lock contention because the data in the same locking entity cannot be accessed concurrently when that entity is locked. It has been shown in [Cev88] that lock conflict causes more serious overhead than the physical memory access conflict in certain cases. This is because each resolution for a lock conflict takes at least a lock duration time which is perhaps much longer than one remote memory access time.

Each lock duration time may be reduced by using finer locking granularity. However, lock conflict for critical shared variables, if exist, cannot be avoided. Unfortunately, most access structures have critical parts which frequently need to be locked, e.g., root node in the tree type index and directory size dependent variables in dynamic hashing.

Based on the above observation, in order to achieve high concurrency multiple independent access structures need to be constructed for each relation to reduce the amount of sharing for critical shared variables. By multiple access structures for each relation, we mean that a set of records pertaining to one relation is partitioned horizontally and each subset of partition contributes to each independent access structure. Thus, we need an appropriate mapping from a set of keys to a set of access structures. Key to access structure mapping determines logical access concurrency while storage allocation scheme for data and access structure determines physical access concurrency.

For many applications we can simplify the general model such that only a group of data which are stored in the same access node, say Q_i, contributes to a unique subset of access structures, say A_j to A_l which are also constructed in the same access node. By this approach storage allocation strategy for data and access structures can be treated integratedly, and the complexity of processing models can also be reduced. Two stage parallel processing model shown in Figure 2.2 represents this idea. In the two stage processing model the first stage corresponds to data storage allocation and the second stage corresponds to key to access structure mapping. The two stage model provides a more systematic design procedure for developing parallel database systems.

2.2. Two Stage Parallel Processing Model for Database Systems

The basic idea of this model is to partition data mapping into two stages. As shown in the figure, the first stage, H1, is called *Data Distribution* algorithm and the second stage, H2, is called *Data Construction* algorithm.

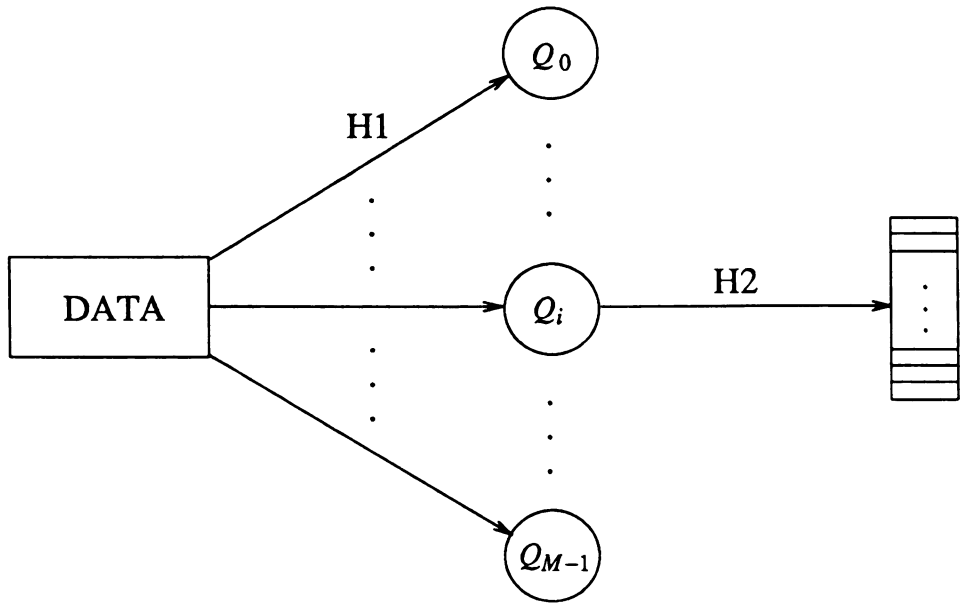


Figure 2.2. Two Stage Parallel Processing Model

The data distribution algorithm, which is the same as data storage allocation in the general model, determines how the data is appropriately distributed to the parallel access nodes so that maximum concurrency is achieved between the access nodes. The data construction algorithm, on the other hand, determines the appropriate data structure to minimize the access time. It receives data from the data distribution algorithm and then create local access structures such as hashed or indexed files.

In general, the following strategies can be employed for data distribution :

(B1) Declustering based on query's data reference pattern

(B2) Random distribution

(B3) Objective specific declustering

(B4) Clustering based on data reference pattern

In method B1, the data distribution technique takes advantage of the data reference pattern of a query. For example, if a query references numerous records, the strategy may be to distribute the data so that these records are stored uniformly among the nodes.

This approach is useful for A1 type applications discussed in chapter 1. In method B2, records are randomly distributed between the nodes. This method is simple, but may not guarantee a good distribution. In the objective specific method, records are allocated to optimize certain objective functions. For example, in [Pra86] Pramanik et al. propose a data distribution technique to construct multiple directories for a single relation, where a record is allocated to the node which has the smallest directory size. It has been shown in that paper that this approach gives the minimum total directory size. However, declustering of data may not be always beneficial. For example, if the communication cost between access nodes is large, clustering may give better performance than declustering. So, B4 type strategies may depend on the interconnection network.

Data distribution algorithms can be a functional mapping which depends only on data values. It maps a set of data values into a set of nodes. For example, if node addresses are determined by hashed values of input data, the distribution algorithm is a mapping which is independent of time or other system parameters. On the other hand, the distribution algorithm need not be functional. For example, random distribution may map the same record to different nodes at different time. Since data accesses are content-based for most database applications, it is advantageous to make a data distribution algorithm functional depending only on data values.

Let D be a set of data and Z_M = {0, 1, . . . , M-1} be a set of parallel access nodes. Let a data distribution algorithm be a function from D to Z_M. Since actual data are usually unevenly distributed in the data domain, data distribution algorithms are commonly designed based on the hashed values of data which are evenly distributed in hashed address space. Thus, we define data distribution algorithm H1 as a composition of two functions, $H1^{(1)}$ and $H1^{(2)}$, such that $H1^{(1)}$ is a mapping from D to T and $H1^{(2)}$ is a mapping from T to Z_M, where T is the set of hashed values. Figure 2.3 shows two level implementation of H1 for the abstract model in Figure 2.2. This model can be thought of as one class of the two stage model.

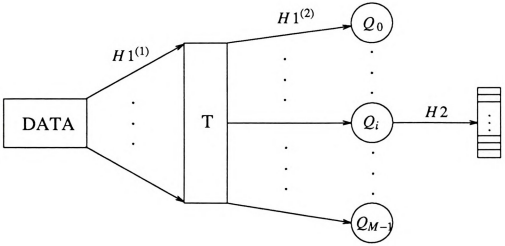


Figure 2.3. Two Level Implementation of H1 for The Two Stage Model

Let H2 be a hash-based data construction algorithm and LD be a set of entries in all local directories generated by H2 for a given file system. If there exists one-to-one correspondence between T and LD, T is called a *real global directory*. Otherwise, it is called a *virtual global directory*.

When T is a real global directory, the set of all the local directories can be thought of as a partition of T. $H1^{(1)}$ is usually static because the use of dynamic hashing for $H1^{(1)}$ will cause significant overhead due to internode data movement. However, static hashing scheme for $H1^{(1)}$ may result in very sparse local directories or long overflow chains. These problems can be avoided by using a virtual global directory, where the actual local directory is determined by H2.

When T is a real global directory, the ratio |T|/|D| directly affects the data retrieval time as well as storage utilization. On the other hand, it is more flexible that T is used as a virtual global directory. The comparison of these two approaches will be described in more detail in chapter 3. Functional distribution, and real/virtual global directory concepts are used for parallel processing of partial match queries and range queries presented in chapter 3 and 4.

CHAPTER 3

PARALLEL PROCESSING STRATEGIES FOR PARTIAL MATCH QUERIES

3.1. Introduction

Many information processing systems involve the application which accesses the records in a file having the values of a specific set of attributes in common. A file is a collection of records each of which is defined as an ordered n-tuple $(r_0, \ldots, r_{n-1})$ of n values which are the keys or attributes of the records. When a query is allowed to specify conditions on more than one attributes, the search performed for this query is called *multikey searching*, or *associative searching*. Multikey searching is used in queries such as partial match retrieval and multiattribute range queries. A partial match query is a query where some of the attributes are specified and the rest of them are unspecified. For example, query q = [Age = *, Department = "mathematics", State = "Ohio"] is a partial match query, where * denotes don't care condition. Since each attribute in a file may or may not be specified in a partial match query, a set of records need to be retrieved. When a file is constructed on parallel devices, it is important to store these records to maximize concurrency. In this chapter we investigate parallel processing strategies for partial match queries. Optimal data distribution methods and appropriate data construction methods are described.

It has been shown in [Rot74] that multikey hashing is effective for partial match retrieval type applications. Multikey hash function, H, for a file consisting of n fields is an ordered n functions $<h_1, \cdots, h_n>$ such that given a record $r = <r_1, \cdots, r_n>$, $H(r) = <h_1(r_1), \cdots, h_n(r_n)>$. $H(r)$ is called a bucket. Rivest [Riv76] and Rothnie, et al. [Rot74] have independently proposed the use of multikey hashing, as an alternative to

inverted files, to reduce the total search time for partial match retrieval type queries. The design of multikey hash functions was considered in [Bur76]. The determination of each field size for minimum search time based on query statistics was also investigated by [Aho79, Bol79]. In [4] it has been shown that the problem of finding the optimal field sizes for multikey hashing scheme is NP-hard. The main focus of those research is on minimizing the total number of bucket accesses. Our objective in this chapter is to achieve maximum parallelism by distributing buckets in multikey hashing.

There are a few heuristic methods for distributing data in partial match retrieval type queries. Du, et al. have proposed Disk Modulo (DM) distribution method [DuS82]. The DM distribution method is simple and elegant but does not work well in many cases. For example, it may not give optimal distribution if some of the field sizes are less than the given number of devices. So, for a large number of parallel access nodes, the DM distribution method may not be appropriate. Generalized Disk Modulo (GDM) method has also been proposed in [DuS82] to overcome this problem. This method gives a sufficient condition to achieve optimal distribution. However, no general method has been given to find the optimal distribution parameters. In fact, the problem of finding the optimal parameter values could be very complex [DuS82]. Since DM distribution method does not work well for binary cartesian product file (binary cartesian product file is a cartesian product file in which each field contains only two elements), several heuristics have been proposed by [Du86, Sun85] to distribute buckets in binary cartesian product files. These heuristics are also special cases of the GDM distribution method. Several useful properties of these modulo based distribution methods have also been given in [Sun87]. Other approaches such as data distribution methods based on minimal spanning trees and short spanning paths have been proposed in [Fan86].

We propose Fieldwise eXclusive-or (FX) distribution methods which give better performance for a wider range of partial match queries than existing methods. The main

idea of FX distribution methods is the use of bitwise exclusive-or operation on the field values which are computed by multikey hashing. Here, we show several useful characteristics of exclusive-or operation for optimal data distribution. Field transformation techniques have been used to increase the scope of optimality in FX distribution methods.

The rest of this chapter is organized as follows. In section 3.2, we describe definitions and terminology. In section 3.3, we define Basic FX distribution method and its optimality conditions. In section 3.4, we present field transformation techniques to increase the scope of optimality over the Basic FX distribution method. The extended optimality conditions for these field transformation schemes are also described. In section 3.5, we compare the performance of FX distribution methods with those of the other distribution methods proposed in the past. We discuss efficient data construction methods in section 3.6.

3.2. Definitions and Terminology

Before describing FX distribution method, it is necessary to introduce some notations as well as relevant definitions and assumptions.

Definition 3.1.

(a) $f_i = \{0, 1, \ldots, F_i-1\}$, a set of hashed values of field i by the i-th hashing function in multikey hashing.

(b) F_i denotes $|f_i|$.

(c) M denotes the number of parallel devices.

(d) N is the set of all natural numbers including 0.

(e) Z_M is the set of all integers from 0 to M-1.

(f) $(a_{m-1} \ldots a_0)_B$ is a binary notation of an integer, where a_i is a binary digit.

$|f_i|$ is assumed to be a power of 2 which is common for hash directory files for partitioned hashing schemes [Aho79]. The number of devices M is also assumed to be a power of 2. Z_M will be frequently used to denote the set of devices.

Definition 3.2. Let $f_1 \times f_2 \times \ldots \times f_n$ be the set of all buckets, where $\times$ denotes the cartesian product of sets. When the given number of devices is M, data distribution method is a function from $f_1 \times f_2 \times \ldots \times f_n$ to Z_M.

Definition 3.3. Let R(q) be the set of buckets which satisfy qualifications for a partial match query q. The distribution method is called *strict optimal* for a partial match query in a given file system if each device has no more than $\lceil |R(q)|/M \rceil$ number of buckets.

Definition 3.4. When the distribution method is strict optimal for all possible partial match queries in a given file system, it is called *perfect optimal* for that file system.

Definition 3.5. The distribution method is called *k-optimal*, $0 \leq k \leq n$, for a given file system consisting of n fields, when it is strict optimal for all partial match queries which have exactly k unspecified fields.

Thus, the distribution method is perfect optimal, if it is k-optimal for all $k = 0, \ldots, n$. Note that some authors exclude cases where the number of unspecified fields is 0 (i.e., exact match) and the number of unspecified fields is n (i.e., retrieval of whole file) from partial match queries.

Definition 3.6. [+] denotes exclusive-or operation between two bits. We will use the same notation [+] to denote exclusive-or operation between integers and sets of integers as follows. When $X = (a_{m-1} \ldots a_0)_B$ and $Y = (b_{m-1} \ldots b_0)_B$ are two integers, $X [+] Y = (a_{m-1} [+] b_{m-1} \ldots a_0 [+] b_0)_B$. If X is an integer and $Y = \{y_1, \ldots, y_L\}$ is a set of integers, X [+] Y is defined as $\{X [+] y_i \mid y_i \in Y\}$. If both $X = \{x_1, \ldots, x_K\}$ and $Y = \{y_1, \ldots, y_L\}$ are sets of integers, X [+] Y is defined as $\{x_i [+] y_j \mid x_i \in X, y_j \in Y\}$

For example, if $X_1 = 2$ and $Y_1 = 3$, then X_1 [+] $Y_1 = 1$. If $X_2 = 2$ and $Y_2 = \{0, 1, 2, 3\}$, then X_2 [+] $Y_2 = \{0, 1, 2, 3\}$.

Definition 3.7. $\overset{n}{\underset{i=1}{[+]}}(Y_i) = Y_1$ [+] Y_2 [+] Y_3 $\cdots$ [+] Y_n.

Note that [+] operator is associative and $\overset{n}{\underset{i=1}{[+]}}$ is a shorthand notation for performing exclusive-or operation between sets of integers $Y_1, Y_2, \ldots, Y_n$.

We assume that precedence of multiplicative operator * is higher than that of exclusive-or operator [+], which means that in the absence of parentheses, multiplication is done before exclusive-or operation. But the precedence of [+] is assumed to be higher than that of + (addition). We will leave out the multiplication operator * whenever there is no ambiguity (e.g., AB instead of A*B).

When d is a power of 2, and $J_1, J_2 \in$ N, we will use the following relations implicitly between exclusive-or operator and arithmetic operators + and *.

(1) If $J_2 <$ d, then $J_1 d$ [+] $J_2 = J_1 d + J_2$

(2) $J_1 d$ [+] $J_2 d = (J_1$ [+] $J_2)d$

(3) $J_1 d$ [+] $J_2 d = 0$ if and only if J_1 [+] $J_2 = 0$

(4) If $J_1 <$ d and $J_2 <$ d, then J_1 [+] $J_2 <$ d

(5) $(J_1$ [+] $J_2)$ *mod* $d = (J_1$ *mod* $d)$ [+] $(J_2$ *mod* $d)$

Since the proof is straightforward, we leave out the proof. Note that if d is not a power of 2, the above relations may not hold.

Definition 3.8. Let $(a_{s-1} \ldots a_{m-1} \ldots a_0)_B$ be the binary notation of an integer l, where $m = \log_2 M$. We define T_M : N $\rightarrow Z_M$ as a function such that $T_M(l) = (a_{m-1} \ldots a_0)_B$.

Thus, function T_M returns only the rightmost $\log_2 M$ bits of domain values. Since we can add arbitrarily large number of 0's to the left of the given binary number of an

integer l, function T_M is always defined for any power of 2 integer M. It is also easy to see that for any $J_1, J_2 \in$ N, $T_M(J_1\ [+]\ J_2) = T_M(T_M(J_1)\ [+]\ J_2) = T_M(J_1)\ [+]\ T_M(J_2)$.

3.3. Basic FX Distribution Method

Let $f_1 \times f_2 \times \ldots \times f_n$ be a set of all buckets. The Basic FX distribution method allocates bucket $<J_1, \ldots, J_n>$ into device $T_M \left[\underset{j=1}{\overset{n}{[+]}} (J_j) \right]$, where $J_j \in f_j$ for $j = 1, \ldots, n$.

Example 3.1. Figure 3.1 shows the bucket distribution by the Basic FX distribution method, where $f_1 = \{0, 1\}$, $f_2 = \{0, 1, 2, 3, 4, 5, 6, 7\}$ and M = 4. In the figure, binary numbers are used for field values and decimal numbers are used for Device No. (This convention will be used in all the examples of FX distribution). Here, Device No = $T_M \left[J_1\ [+]\ J_2 \right]$, where $J_1 \in f_1$, $J_2 \in f_2$ and T_M returns the rightmost two bits of the result of $J_1\ [+]\ J_2$.

f_1	f_2	Device No
000	000	0
000	001	1
000	010	2
000	011	3
000	100	0
000	101	1
000	110	2
000	111	3
001	000	1
001	001	0
001	010	3
001	011	2
001	100	1
001	101	0
001	110	3
001	111	2

Figure 3.1. Basic FX Distribution

As shown in Figure 3.1, the Basic FX distribution method is strict optimal for any partial match query in this file system. For example, if f_1 value is $(001)_B$ and f_2 is unspecified, then we have to access eight buckets $<(001)_B,(000)_B>$, ... , $<(001)_B,(111)_B>$. Since each device has two qualified buckets for this partial match query, the Basic FX distribution method is strict optimal for this query.

Lemma 3.1. Z_M is a set which contains M different nonnegative integers from 0 to M-1. Let k be some integer such that $0 \le k \le M-1$. Then Z_M [+] k = Z_M.

Proof: Clearly, for any nonnegative integer A_a which is less than M, $0 \le k$ [+] $A_a \le M-1$. Thus, it is sufficient to show that for any two different elements of Z_M, the results of exclusive-or with k are different. Let $A_a = (a_{m-1}a_{m-2} \cdots a_0)_B$ and $A_b = (b_{m-1}b_{m-2} \cdots b_0)_B$, where $m = \log_2 M$ and at least one $a_i \neq b_i$, i.e., $A_a \neq A_b$. Let A_a [+] k = $(c_{m-1} \cdots c_0)_B$ and A_b [+] k = $(d_{m-1} \cdots d_0)_B$. Assume A_a [+] k = A_b [+] k. Then $c_j = d_j$ for all $j = 0, \ldots, m-1$. Since $c_j = d_j$ implies $a_j = b_j$, it follows that $a_j = b_j$ for all j = 0, ... , m-1. This contradicts the assumption of $A_a \neq A_b$. Since $0 \le k$ [+] $A_i \le M-1$ for all $A_i \in Z_M$ and k [+] $A_a \neq$ k [+] A_b for any two different A_a and A_b in Z_M, Z_M [+] k is the same as Z_M. □

Example 3.2. Let Z_8 = {0, 1, 2, 3, 4, 5, 6, 7} and k = 3. Then Z_8 [+] k = {3, 2, 1, 0, 7, 6, 5, 4} = Z_8.

In the proof of Lemma 3.1, it is shown that for any two different nonnegative integers I, J and any nonnegative integer k, I [+] k is different from J [+] k. We call this *XOR uniqueness* property.

We can observe that the property described in Lemma 3.1 (this also implies XOR uniqueness property) is very useful for optimal file distribution of partial match retrieval type applications. This property is fundamental, and will be used in various places of this chapter to develop further techniques for optimal distribution.

In the following Theorem 3.1 and Theorem 3.2 we will describe the types of partial match queries for which the Basic FX distribution method gives optimal distribution.

Theorem 3.1. The Basic FX distribution method is always 0-optimal and 1-optimal.

Proof: (1) 0-optimal : This is trivially true.

(2) 1-optimal : Let only one field i be unspecified and $T_M\left(\underset{j\neq i}{[+]}(J_j)\right)$ = h, where J_j is the specified value of field j. Thus, h gives the projection of the rightmost $\log_2 M$ bits of the value obtained by the exclusive-or of all the specified values of the query. There are two cases, $F_i \leq M$ and $F_i > M$. When $F_i \leq$ M, for all I $\in f_i$, $T_M(I)$ [+] h is different from each other by XOR uniqueness property. Therefore, the distribution is optimal. (Note that $T_M(A\,[+]B) = T_M(A)\,[+]\,T_M(B) = T_M(A\,[+]\,T_M(B)) = T_M(T_M(A)\,[+]\,T_M(B))$). When $F_i >$ M, let F_i = A*M. By Lemma 3.1 f_i [+] h = Z_{A*M}. Since $\#\{\alpha \in Z_{A*M} \mid T_M(\alpha) = z\}$ = A for any $z \in Z_M$, the Basic FX distribution method allocates A number of buckets to each device. (Note that # denotes the cardinality of a set.) Therefore, the distribution is optimal. □

Theorem 3.1 says that the Basic FX distribution method is strict optimal for any partial match query in which the number of unspecified fields is 0 or 1. Note that in the above expression, $\#\{\alpha \in Z_{A*M} \mid T_M(\alpha) = z\}$ = A, A denotes the number of qualified buckets that correspond to a particular device z.

Theorem 3.2. For any partial match query which has two or more unspecified fields, the Basic FX distribution method is strict optimal, if there exists at least one unspecified field i such that $F_i \geq M$.

Proof: For partial match query q, let q(f) = $\{i_1, i_2, \ldots, i_k\}$ be the set of unspecified fields in which the size of at least one of these fields is greater than or equal to M. Without loss of generality, assume $F_{i_1} \geq M$, i.e., $F_{i_1} = A_{i_1} * M$ for some nonnegative integer A_{i_1}. Let h = $T_M\left(\underset{j \notin q(f)}{[+]}(J_j)\right)$, where J_j denotes the specified value of field j

which is not in q(f). Thus, h is the projection of the rightmost $\log_2 M$ bits of the value obtained by the exclusive-or of all the specified field values of the query. By Lemma 3.1, $h\ [+]\ f_{i_1} = Z_{A_{i_1} * M}$ and hence $\#\{J_{i_1} \in f_{i_1} \mid T_M(h\ [+]\ J_{i_1}) = z\} = A_{i_1}$ for all $z \in Z_M$. Here, we have done exclusive-or of h with the set of values of the unspecified field i_1. A_{i_1} gives the number of unique values of the unspecified field i_1 that correspond to a particular value z. Now we will exclusive-or the set of values $h\ [+]\ f_{i_1}$ with a value of the unspecified field i_2. Let $J_{i_2} \in f_{i_2}$. By Lemma 3.1 $\#\{J_{i_1} \in f_{i_1} \mid T_M(h\ [+]\ J_{i_1}\ [+]\ J_{i_2}) = z\} = A_{i_1}$ for all $z \in Z_M$. Note that this will not change the number of unique values of field i_1 that correspond to a particular z. We also used the property, $T_M(h\ [+]\ J_{i_1}\ [+]\ J_{i_2}) = T_M(h\ [+]\ J_{i_1}\ [+]\ T_M(J_{i_2}))$. Thus, $\#\{(J_{i_1}, J_{i_2}) \in f_{i_1} \times f_{i_2} \mid T_M(h\ [+]\ J_{i_1}\ [+]\ J_{i_2}) = z\} = A_{i_1} F_{i_2}$ for all $z \in Z_M$. Here, the number of unique values for each z is more by a factor F_{i_2} because the size of the domain has increased by a factor F_{i_2}. By continuing this argument, $\#\{(J_{i_1}, \cdots J_{i_k}) \in f_{i_1} \times \cdots \times f_{i_k} \mid T_M\left(h\ [+]\ \left[\underset{p=1}{\overset{k}{[+]}}(J_{i_p})\right]\right) = z\} = A_{i_1} \prod_{p=2}^{k} F_{i_p} = (\prod_{p=1}^{k} F_{i_p}) / M$ for all $z \in Z_M$.

□

Note that Theorem 3.1 works for partial match queries with zero or one unspecified field while Theorem 3.2 applies to partial match queries with more than one unspecified fields.

Corollary 3.1. When all field sizes are no less than the given number of devices M, the Basic FX distribution method is perfect optimal.

Proof: This is a direct consequence of Theorem 3.2. □

Theorem 3.1 and 3.2 show general characteristics of exclusive-or operation for optimal distribution. However, the Basic FX distribution method does not give optimal distribution for partial match queries with 2 or more unspecified fields, when the size of none of the unspecified fields is greater than or equal to M. For example, when M = 16

and all others are the same as in Example 3.1, the distribution is not optimal if both fields are unspecified. Proposition 3.1 gives the conditions for optimal distribution for these cases.

Proposition 3.1. Let q(f) = $\{i_1, i_2, \ldots, i_k\}$ be the set of unspecified fields for partial match query q, where $F_j < M$, for all $j \in$ q(f). FX distribution is strict optimal for partial match query q, if there exist a set of fields $\{i_1, \ldots, i_j\} \subseteq$ q(f) such that $|f_{i_1} \times \cdots \times f_{i_j}| \geq M$ and $\#\{(J_{i_1}, \cdots J_{i_j}) \in f_{i_1} \times \cdots \times f_{i_j} \mid T_M\left[\underset{p=1}{\overset{j}{[+]}}(J_{i_p})\right] = z\} = |f_{i_1} \times \cdots \times f_{i_j}|/M$ for all $z \in Z_M$.

Proof: Let $F_{i_1} \times \cdots \times F_{i_j} = A_{i_j} * M$, $A_{i_j} \in$ N, $A_{i_j} \geq 1$. Let $h = T_M\left[\underset{l \in q(f)}{[+]}(J_l)\right]$, where J_l denotes the specified value of field l. The remainder of the proof is similar to that of Theorem 3.2. □

Proposition 3.1 says that even though the sizes of all the unspecified fields are less than the given number of devices M, we can still guarantee optimal distribution, if (1) there exists a subset of the unspecified fields, the size of whose cartesian product is greater than or equal to M and (2) the records projected on these sets of fields are distributed uniformly among the M devices. In other words, optimal distribution for a subset of fields guarantees strict optimal distribution for many queries in which those fields are unspecified.

However, when the size of none of the unspecified fields is greater than or equal to M, the conditions given in Proposition 3.1 are not satisfied in the Basic FX distribution method. Thus, in the next section we propose field transformation techniques for the fields whose sizes are less than the given number of devices M. These field transformation techniques increase the scope of optimality by themselves, and can also utilize Proposition 3.1. The following paragraph exemplifies the idea of field transformation techniques.

Let $f_1 = \{0, 1\}$, $f_2 = \{0, 1, 2, 3, 4, 5, 6, 7\}$ and M = 16. As we discussed, the Basic FX distribution method is not perfect optimal for this file system. Let X be an one-to-one mapping such that $X(f_1) = \{0, 8\}$. When the Basic FX distribution method is applied for the set of ordered 2-tuples, $X(f_1) \times f_2$, the distribution is perfect optimal. (It can be easily verified by substituting $(1000)_B$ for $(001)_B$ in f_1 column of Figure 3.1.) Now, our objective is to find a general one-to-one mapping, X, such that the Basic FX distribution method for $X(f_1) \times f_2$ gives optimal distribution. It will be shown that for any values of $|f_1|$, $|f_2|$ and M, such mapping can be easily found.

We will present several field transformation techniques which give the mapping, X, described above. Even though the techniques developed here may not achieve perfect optimal distribution in all the cases, this extended FX distribution method gives strict optimal distribution for a large class of partial match query.

3.4. Extended FX Distribution Methods

In the previous section the Basic FX distribution method was described. In this section we extend the Basic FX distribution method by using field transformation techniques.

Let $f_1 \times f_2 \times \ldots \times f_n$ be a set of all buckets. Extended FX distribution methods (the Basic FX distribution method with field transformation techniques) allocates bucket $\langle J_1, \ldots, J_n \rangle$, $J_j \in f_j$ for $j = 1, \ldots, n$, into device $T_M \left[\left[\sum_{j=1}^{n} \right] (X_j(J_j)) \right]$, where

i) if $|f_j| \geq M$, X_j is the identity function,

ii) if $|f_j| < M$, X_j is an element of set of injective (one-to-one) functions whose domains are f_j and ranges are Z_M.

X_j is called a field transformation function.

When X_j is the identity function for all $j = 1, \ldots, n$, Extended FX distribution methods reduce to the Basic FX distribution method. It is easy to see that all the

lemmas and theorems that hold for the Basic FX distribution method also hold for Extended FX distribution methods.

By the above definition, a field transformation function can be any one-to-one function. However, we are only interested in the field transformation functions by which we can achieve strict optimal distribution for various types of partial match queries for which the Basic FX distribution method does not give strict optimal distribution.

Since the fields whose sizes are no less than the given number of devices M, do not cause any problem (whether it is specified or not), in this section we will focus only on the fields whose sizes are less than M. From now on, we will simply call FX distribution methods instead of Extended FX distribution methods.

In the following subsections we describe field transformation functions which are used for the fields whose sizes are less than the given number of devices.

3.4.1. Field Transformation Functions for Partial Match Queries

The field transformation functions developed for FX distribution methods are as follows.

Definition 3.9. Let $f_l = \{0, \ldots, F_l-1\}$, and let $|f_l|$ and M are power of 2.

(1) $I : \mathrm{N} \rightarrow \mathrm{N}$ is an identity function.

(2) When $|f_l| < \mathrm{M}$, $\mathbf{U}^{M,|f_l|} : f_l \rightarrow Z_M$ is a function such that $\mathbf{U}^{M,|f_l|}(l) = ld^{M,|f_l|}$, where $d^{M,|f_l|} = M/|f_l|$.

(3) When $|f_l| < \mathrm{M}$, for $x = 1, \ldots, y$ in which y is a maximum integer such that $|f_l|^y < M$, $\mathbf{IU_x}^{M,|f_l|} : f_l \rightarrow Z_M$ is a function such that $\mathbf{IU_x}^{M,|f_l|}(l) = l\ [+]\ \left[\underset{k=1}{\overset{x}{[+]}} ld_k^{M,|f_l|}\right]$ where $d_k^{M,|f_l|} = M/|f_l|^k$.

The function $IU_x^{M,|f_l|}$ is a general notation for functions $IU_1^{M,|f_l|}$, $IU_2^{M,|f_l|}$, . . . , etc. For example, functions $IU_1^{M,|f_l|}$ and $IU_2^{M,|f_l|}$ are defined as follows :

(a) When $|f_l| < M$, $\mathbf{IU_1}^{M,|f_l|} : f_l \rightarrow Z_M$ is a function such that $\mathbf{IU_1}^{M,|f_l|}(l) = l\ [+]\ ld^{M,|f_l|}$, where $d^{M,|f_l|} = M/|f_l|$.

(b) When $|f_l|^2 < M$, $\mathbf{IU_2}^{M,|f_l|} : f_l \rightarrow Z_M$ is a function such that $\mathbf{IU_2}^{M,|f_l|}(l) = l\ [+]\ ld_1^{M,|f_l|}\ [+]\ ld_2^{M,|f_l|}$, where $d_1^{M,|f_l|} = M/|f_l|$, $d_2^{M,|f_l|} = M/|f_l|^2$.

Example 3.3. Let $f_1 = \{0, 1, 2, 3\}$ and $f_2 = \{0, 1\}$ and M = 16.

(a) $U^{16,4}(f_1) = \{0, 4, 8, 12\}$, and $IU_1{}^{16,4}(f_1) = \{0[+]0, 1[+]4, 2[+]8, 3[+]12\} = \{0, 5, 10, 15\}$.

(b) $IU_2{}^{16,2}(f_2) = \{0, 1[+]8[+]4\} = \{0, 13\}$ and $IU_3{}^{16,2}(f_2) = \{0, 1[+]8[+]4[+]2\} = \{0, 15\}$.

We have defined basic transformation functions, I, $U^{M,|f_l|}$, $IU_1{}^{M,|f_l|}$, ... , $IU_x{}^{M,|f_l|}$ which will be used in various combinations for optimal data distribution. For example, for any values of $|f_i|$, $|f_j|$ and M, it will be shown that FX distribution methods distribute ordered 2-tuples in $I(f_i) \times U^{M,|f_j|}(f_j)$ optimally.

When $X_1{}^{M1,|f_i|}$ and $X_2{}^{M2,|f_j|}$ functions are applied to fields *i*, and *j*, two *transformation methods are said to be the same*, if $X_1 = X_2$. For example, $U^{M1,|f_i|}$ and $U^{M2,|f_j|}$ are said to be the same transformation methods. But $U^{M1,|f_i|}$ and $IU_1{}^{M2,|f_j|}$, or $IU_1{}^{M1,|f_i|}$ and $IU_2{}^{M2,|f_j|}$ are said to be *different transformation methods*.

Because of notational complexity, we will leave out the superscripts M, $|f_l|$ from transformation functions and their parameters whenever there is no ambiguity.

It is easy to see that for any proper subset f_l of Z_M where $|f_l|$ is some power of 2, U transformation function satisfies the requirement of field transformation functions (i.e., one-to-one function whose domain is f_l and range is Z_M). We will show later that IU_1, . . . , IU_x transformation functions also satisfy this requirement. From the definition of U

transformation function we see that the transformed domain elements are equally spaced. In other words, when they are linearly ordered, any two adjacent elements have the same distance. Several useful properties of $IU_1, \ldots, IU_x$ transformation functions will be described in Lemma 3.7, Lemma 3.9, Lemma 3.11 and Lemma 3.12.

In the following sections, we will present various combinations of field transformation functions by which FX distribution methods give optimal distribution. Note that we focus only on the fields whose sizes are less than the given number of devices.

3.4.2. I and U Field Transformation Functions

In this section we show in Theorem 3.3 that for any values of F_i, F_j and M, FX distribution methods distribute ordered 2-tuples in $I(f_i) \times U(f_j)$ optimally. Theorem 3.3 uses Lemma 3.2 which is an extended version of Lemma 3.1.

Lemma 3.2. For a nonnegative integer L, let $L = aw + b$, where w is a power of 2, and a, b are some nonnegative integers such that $0 \le b \le w\text{-}1$. When $Z_w = \{0, 1, \ldots, w\text{-}1\}$, Z_w [+] $L = \{aw, aw+1, \ldots, (a+1)w\text{-}1\}$.

Proof: Let $(l_{m-1} \cdots l_k\ l_{k-1} \cdots l_0)_B$ be a binary notation of L, where l_k has weight w. Then, $(l_{m-1} \ldots l_k)_B = a$. All elements in Z_w has the form $(0 \ldots 0\ b_{k-1} \cdots b_0)_B$. When L [+] α for any $\alpha \in Z_w$, the result should have the form $(l_{m-1} \cdots l_k\ c_{k-1} \cdots c_0)_B$ which is $aw + (c_{k-1} \cdots c_0)_B$, where $(c_{k-1} \cdots c_0)_B$ is equal to $(l_{k-1} \cdots l_0)_B$ [+] $(b_{k-1} \cdots b_0)_B$. Thus the proof follows by Lemma 3.1. □

Theorem 3.3. When there are only two fields i, j whose sizes are less than the given number of devices M, the FX distribution method with $I(f_i)$ and $U(f_j)$ is perfect optimal.

Proof: There are two cases, i.e., $F_iF_j < M$ and $F_iF_j \ge M$. Let $d_j = M/F_j$.

(case 1) $F_iF_j < M$ (i.e., $d_j > F_i$)

$I(f_i)$ [+] $U(f_j) = \{0, 1, \ldots, F_i\text{-}1\} \cup \{d_j, d_j+1, \ldots, d_j+F_i\text{-}1\} \cup \ldots \{M\text{-}d_j, M\text{-}d_j+1, \ldots,$

M-d_j+F_i-1} by Lemma 3.2. In the right hand side all sets are disjoint and largest element is less than M because $d_j > F_i$. So, for all $z \in Z_M$, $\#\{(J_i,J_j) \in f_i \times f_j \mid I(J_i)\ [+]\ U(J_j) = z\} \le 1$. Therefore, it is 2-optimal. 0 and 1-optimal come from Theorem 3.1.

(case 2) $F_iF_j \ge M$ (i.e., $d_j \le F_i$)

Let $F_iF_j = A*M$ (i.e., $d_j = F_i/A$). In order to be 2-optimal, for all $z \in Z_M$, $\#\{(J_i,J_j) \in f_i \times f_j \mid I(J_i)\ [+]\ U(J_j) = z\}$ should be equal to A.

(1) $0 \le U(J_j) \le (A\text{-}1)d_j$

For each $U(J_j)$ in this range, $I(f_i)\ [+]\ U(J_j) = \{0, 1, \ldots, F_i\text{-}1\}$ by Lemma 3.2. Let this set be S_0. Since there are A number of such $U(J_j)$ in this range, for all $s \in S_0$ $\#\{(J_i,J_j) \in f_i \times f_j \mid 0 \le U(J_j) \le (A-1)d_j,\ I(J_i)\ [+]\ U(J_j) = s\} = A$.

(2) $Ad_j \le U(J_j) \le (2A\text{-}1)d_j$

For each $U(J_j)$ in this range, $I(f_i)\ [+]\ U(J_j) = \{F_i, F_i+1, \cdots, 2F_i-1\}$ by Lemma 3.2. Let this set be S_1. Since there are A number of such $U(J_j)$ in this range, for all $s \in S_1$ $\#\{(J_i,J_j) \in f_i \times f_j \mid Ad_j \le U(J_j) \le (2A-1)d_j,\ I(J_i)\ [+]\ U(J_j) = s\} = A$.

-

-

(M/F_i) $(M/F_i-1)Ad_j \le U(J_j) \le M\text{-}d_j$

For each $U(J_j)$ in this range, $I(f_i)\ [+]\ U(J_j) = \{(M/F_i-1)F_i, (M/F_i-1)F_i+1, \ldots . M-1\}$. Let this set be S_{M/F_i-1}. Since there are A number of such $U(J_j)$ in this range, for all $s \in S_{M/F_i-1}$, $\#\{(J_i,J_j) \in f_i \times f_j \mid (M/F_i-1)Ad_j \le U(J_j) \le M-d_j,\ I(J_i)\ [+]\ U(J_j) = s\} = A$. Since $\bigcup_{p=0}^{M/F_i-1} S_p = Z_M$ and there are A repetitions for each element in Z_M through $I(f_i)\ [+]\ U(J_j)$, $J_j = 0, \cdots, F_j-1$, it is 2-optimal. 0 and 1-optimal come from Theorem 3.1. □

Example 3.4. Let $f_1 = \{0, 1, 2, 3\}$, $f_2 = \{0, 1, 2, 3\}$ and M = 16. Figure 3.2 shows the bucket distribution by FX and DM distribution methods. Note that $I(f_1) = \{0, 1, 2, 3\}$

and $U(f_2)$ = {0, 4, 8, 12} are I and U transformed values of f_1, f_2 and denoted by binary numbers. Here, Device No = $T_M(I(J_1)$ [+] $U(J_2))$ for FX distribution methods, and Device No = $(J_1 + J_2)$ *mod* 16 for the DM distribution method, where $J_1 \in f_1$, $J_2 \in f_2$.

f_1	f_2	$I(f_1)$	$U(f_2)$	Device No (FX)	Device No (DM)
0	0	0000	0000	0	0
0	1	0000	0100	4	1
0	2	0000	1000	8	2
0	3	0000	1100	12	3
1	0	0001	0000	1	1
1	1	0001	0100	5	2
1	2	0001	1000	9	3
1	3	0001	1100	13	4
2	0	0010	0000	2	2
2	1	0010	0100	6	3
2	2	0010	1000	10	4
2	3	0010	1100	14	5
3	0	0011	0000	3	3
3	1	0011	0100	7	4
3	2	0011	1000	11	5
3	3	0011	1100	15	6

Figure 3.2. FX Distribution with I And U Transformation

The FX distribution in Figure 3.2 is perfect optimal. But in DM distribution, the distribution is skewed. The GDM method can also give optimal distribution by multiplying 3 to the first field values and by 4 to the second field values. However, these parameters should be found by trial and error. On the other hand, FX distribution techniques give a specific method.

3.4.3. I and IU_x Field Transformation Functions

In this section we show in Theorem 3.4 that for any values of F_i, F_k, and the given number of devices M, FX distribution methods distribute ordered 2-tuples in $I(f_i) \times IU_x(f_k)$ optimally. Lemma 3.3 shows IU_x transformation functions are injective.

Lemma 3.7 shows a useful property of IU_x transformation functions which is used to prove Theorem 3.4. Lemma 3.7 is derived based on Lemma 3.4, 5.3 and 5.4.

Lemma 3.3. Let $f_k = \{0, \ldots, F_k-1\}$ such that $F_k{}^x < M$ for some positive integer x. Then, $IU_x(f_k)$ is a set of F_k elements between 0 and M-1 (i.e., $IU_x(f_k)$ is an injective function).

Proof: (case 1) $x = 1$.

Assume otherwise. Let $d_1 = M/F_k$. Then there exist two different elements $K_1, K_2 \in f_k$ such that K_1 [+] $K_1 d_1 = K_2$ [+] $K_2 d_1$. Let $K' = K_1$ [+] K_2. (Note that for any two different nonnegative integers K_1 and K_2 there always exists positive integer K′ such that $K' = K_1$ [+] K_2). By substituting K_1 [+] K′ for K_2 we have $K_1 d_1 = K'$ [+] $K_2 d_1$. Let $K_1 = (0 \ldots 0\ a_r \ldots a_0)_B$, $K_2 = (0 \ldots 0\ b_s \ldots b_0)_B$ and $K' = (0 \ldots 0\ e_t \ldots e_0)_B$, where r, s, t denote leftmost bit positions in which the binary value is "1" in K_1, K_2, K′, respectively. This implies $a_t \neq b_t$ because $K' = K_1$ [+] K_2. Since d_1 is a power of 2, $d_1 = 2^c$ for some positive integer c. Then, $K_1 d_1 = (0 \ldots 0\ a_{r+c} \ldots a_{t+c} \ldots a_c \ldots 0)_B$, $K_2 d_1 = (0 \ldots 0\ b_{s+c} \ldots b_{t+c} \ldots b_c \ldots 0)_B$, where $a_{t+c} \neq b_{t+c}$. Since e_{t+c} is zero, $a_{t+c} \neq e_{t+c}$ [+] b_{t+c}. This contradicts $K_1 d_1 = K'$ [+] $K_2 d_1$.

(case 2) $x > 1$

Assume otherwise. Then there exist two different elements $K_1, K_2 \in f_k$ such that $(K_1$ [+] $K_1 d_1$ [+] $\ldots$ [+] $K_1 d_x) = (K_2$ [+] $K_2 d_1$ [+] $\ldots$ [+] $K_2 d_x)$, where $d_i = M/F_k{}^i$, $i = 1, \ldots, x$. Since $K_1 d_1$ and $K_2 d_1$ are the only ones which can have binary value "1" between bit position $\log_2 d_1$ and $\log_2 M$ (we considered the rightmost bit position to be zero), the only way the above equation can be satisfied is $K_1 = K_2$. Therefore, contradiction. □

Lemma 3.4. When there are only two fields i, k whose sizes are less than the given number of devices M and $F_i \geq F_k$, the FX distribution method with $I(f_i)$ and $IU_1(f_k)$ is perfect optimal.

Proof: There are two cases, i.e., $F_iF_k < M$ and $F_iF_k \geq M$. Let $d_k = M/F_k$ and $d_i = M/F_i$.

(case 1) $F_iF_k < M$ (i.e., $d_k > F_i$)

For all K $\in f_k$,

$$I(f_i)\ [+]\ K\ [+]\ Kd_k = \{0, 1, \cdots, F_i-1\}\ [+]\ Kd_k$$

$$= \{Kd_k, Kd_k+1, \cdots, Kd_k+F_i-1\}$$

The first equality holds by Lemma 3.1, and the second equality holds by Lemma 3.2. Clearly, no element in Z_M is repeated through $I(f_i)\ [+]\ IU_1(K)$, K=0, ... F_k-1 because $d_k > F_i$.

(case 2) $F_iF_k \geq M$ (i.e., $d_k \leq F_i$)

Let $F_iF_k = AM$. Then, $F_i = Ad_k$ and $Ad_id_k = M$. Let K $\in f_k$

i) $0 \leq K < A$ (i.e., $0 \leq Kd_k < F_i$)

For each K within this range,

$$I(f_i)\ [+]\ K\ [+]\ Kd_k = \{0, 1, \cdots, F_i-1\}\ [+]\ Kd_k$$

$$= \{0, 1, \cdots, F_i-1\}$$

by Lemma 3.1. Since there are A number of such K's within this range, there are A repetitions for each element in Z_{F_i} through $I(f_i)\ [+]\ IU_1(K)$, K=0, ... A-1.

ii) $A \leq K < 2A$ (i.e., $F_i \leq Kd_k < 2F_i$)

For each K within this range,

$$I(f_i)\ [+]\ K\ [+]\ Kd_k = \{0, 1, \cdots, F_i-1\}\ [+]\ Kd_k$$

$$= \{F_i, F_i+1, \cdots, 2F_i-1\}$$

by Lemma 3.1 and Lemma 3.2. Since there are A number of such K's within this range, there are A repetitions for each element in $Z_{2F_i} - Z_{F_i}$ through $I(f_i)\ [+]\ IU_1(K)$, K=A, ... 2A-1.

d_i) $(d_i-1)A \le K < d_iA$ (i.e., $(d_i-1)Ad_k \le Kd_k < Ad_id_k$)

For each K within this range,

$$I(f_i)\ [+]\ K\ [+]\ Kd_k = \{0, 1, \cdots, F_i-1\}\ [+]\ Kd_k$$

$$= \{(d_i-1)F_i, (d_i-1)F_i+1, \cdots, d_iF_i-1\}$$

by Lemma 3.1 and Lemma 3.2. Therefore, there are A repetitions for each element in Z_M - Z_{M-F_i}. Since there are A repetitions for each element in Z_M through $I(f_i)\ [+]\ IU_1(K)$, K=0, ... , F_k−1, it is 2-optimal.

0 and 1-optimal come from Theorem 3.1 for both case (1) and case (2). □

Definition 3.10. When F_j < M and d = M/F_j, there are F_j intervals [0, d), [d, 2d), . . . , [M-d, M) from 0 to M with interval size $d = M/F_j$, where "[" and ")" denote "closed" and "open", respectively.

Lemma 3.5. When there are only two fields i and k such that $F_iF_k = M$, FX distribution with $I(f_i)$ and $IU_1(f_k)$ is perfect optimal if and only if there is exactly one element of $IU_1(f_k)$ at each interval from 0 to M with interval size M/F_k.

Proof: Since $F_iF_k = M$, in order to be perfect optimal distribution for all $z \in Z_M$, $\#\{(J_i,J_k) \in f_i \times f_k \mid I(J_i)\ [+]\ IU_1(J_k) = z\}$ should be equal to one.

(1) Only if : Assume otherwise. Let $d_k = M/F_k$ i.e., $d_k = F_i$. Then for some interval $[ld_k, (l+1)d_k)$, there exist two or more $IU_1(f_k)$ elements. Let u and v be such elements in f_k, i.e., $ld_k \le u\ [+]\ ud_k \le (l+1)d_k$ - 1 and $ld_k \le v\ [+]\ vd_k \le (l+1)d_k$ - 1. Then $I(f_i)\ [+]\ u\ [+]\ ud_k = I(f_i)\ [+]\ v\ [+]\ vd_k = \{ld_k, ld_k+1, \ldots, (l+1)d_k-1\}$. The equality holds by Lemma 3.1 and Lemma 3.2. Note that $F_i = d_k$. Clearly, the distribution is not optimal. Therefore, contradiction.

(2) If : Let $S_a = \{aF_i, aF_i+1, \ldots, (a+1)F_i-1\}$. Since $IU_1(f_k)$ =

$\{id_k + c_i \mid i = 0, \cdots, F_k - 1$, where $0 \le c_i \le d_k\}$, $I(f_i)$ [+] $IU_1(f_k) = \bigcup_{a=0}^{F_k-1} S_a = Z_M$ by Lemma 3.2. Therefore, it is perfect optimal. □

Lemma 3.6. For any F_k which is less than M, there is exactly one element of $IU_1(f_k)$ at each interval from 0 to M with interval size M/F_k.

Proof: For any large F_k there always exist M and F_i such that $M > F_i \ge F_k$ and $F_iF_k = M$ (we can always find such F_i and M). For such M, F_i and F_k, $I(f_i)$ [+] $IU_1(f_k)$ gives perfect optimal distribution by Lemma 3.4. For all such F_k, by Lemma 3.5 there is exactly one element of $IU_1(f_k)$ at each interval from 0 to M with interval size M/F_k. Thus the lemma follows □

For example, when $f_1 = \{0, 1, 2, 3\}$, $f_2 = \{0, 1, \ldots 7\}$ and $M = 16$, $IU_1(f_1) = \{0, 5, 10, 15\}$ and $IU_1(f_2) = \{0, 3, 6, 5, 12, 15, 10, 9\}$. We can see that there is exactly one element of $IU_1(f_1)$ and $IU_2(f_2)$ at each interval with size 4 and 2, respectively.

Lemma 3.7. Let $f_j = \{0, \ldots, F_j - 1\}$, where $F_j^x < M$ for some positive integer x. Then, there is exactly one element of $IU_x(f_j)$ at each interval from 0 to M with interval size M/F_j.

Proof: When $x = 1$, this is true by Lemma 3.6. When $x \ge 2$, let $d_i = M/F_j^i$, $i = 1, \ldots, x$. Since $(Jd_2$ [+] $\ldots$ [+] $Jd_x) < d_1$ for all $J \in f_j$, the proof is clear. □

Theorem 3.4. When there are only two fields i, j and the given number of devices is M such that $F_i < M$ and $F_j^x < M$, for some positive integer x, the FX distribution method with $I(f_i)$ and $IU_x(f_j)$ is perfect optimal.

Proof: Let $d_1 = M/F_j$.

(case 1) $F_iF_j \le M$ (i.e., $d_1 \ge F_i$)

By Lemma 3.7, there is exactly one element of $IU_x(f_j)$ at each interval from 0 to M with interval size d_1. Thus, by Lemma 3.2 $I(f_i)$ [+] $IU_x(f_j)$ is a set of all different F_iF_j elements between 0 and M-1.

(case 2) $F_iF_j > M$ (i.e., $d_1 < F_i$)

Let $F_iF_j = AM$. Thus, $F_i = Ad_1$. Let $IU_x(f_j) = \{t_0, t_1, \ldots, t_{F_j-1}\}$, where $t_0 < t_1 < \cdots < t_{F_j-1}$.

(1) $0 \le t_l < Ad_1$ (i.e., $0 \le t_l < F_i$)

For each t_l within this range, $I(f_i)$ [+] $t_l = \{0, 1, \ldots, F_i-1\}$. Since there are A number of such t_l elements within this range by Lemma 3.7, there are A repetitions for each element in Z_{F_i} through $I(f_i)$ [+] t_l, $l = 0, \cdots, A-1$.

(2) $Ad_1 \le t_l < 2Ad_1$ (i.e., $F_i \le t_l < 2F_i$)

For each t_l within this range, $I(f_i)$ [+] $t_l = \{F_i, F_i+1, \ldots, 2F_i-1\}$ by Lemma 3.2. Since there are A number of such t_l elements within this range, there are A repetitions for each element in $Z_{2F_i} - Z_{F_i}$ through $I(f_i)$ [+] t_l, $l = A, \cdots, 2A-1$.

-

-

(M/F_i) $(M/F_i-1)Ad_1 \le t_l < M/F_iAd_2$ (i.e., $(M/F_i-1)F_i \le t_l < M$)

For each t_l within this range, $I(f_i)$ [+] $t_l = \{(M/F_i-1)F_i, (M/F_i-1)F_i+1, \ldots, (M/F_i-1)F_i+F_i-1\}$ by Lemma 3.2. Since there are A number of such t_l elements within this range, there are A repetitions for each element in $Z_M - Z_{(\frac{M}{F_i}-1)F_i}$ through $I(f_i)$ [+] t_l, $l = (\frac{M}{F_i}-1)A, \ldots, M/F_iA-1$.

Since there are A repetitions for each element in Z_M through $I(f_i)$ [+] t_l, $l = 0, \ldots, F_j-1$, it is 2-optimal. 0 and 1-optimal come from Theorem 3.1. □

Example 3.5. Let $f_1 = \{0, 1, 2, 3\}$, $f_2 = \{0, 1, 2, 3\}$ and M = 16. Figure 3.3 shows the bucket distribution by the FX distribution method with $I(f_1)$ and $IU_1(f_2)$. Here, $IU_1(f_2) = \{0, 5, 10, 15\}$ and Device No = $T_M(I(J_1)$ [+] $IU_1(J_2))$, $J_1 \in f_1$, $J_2 \in f_2$. We can see that the distribution of Figure 3.3 is perfect optimal.

Example 3.6. Let $f_1 = \{0, 1, 2, 3, 4, 5, 6, 7\}$, $f_2 = \{0, 1\}$ and M = 16. Figure 3.4 shows the bucket distribution by the FX distribution method with $I(f_1)$ and $IU_2(f_2)$.

$I(f_1)$	$IU_1(f_2)$	Device No
0000	0000	0
0000	0101	5
0000	1010	10
0000	1111	15
0001	0000	1
0001	0101	4
0001	1010	11
0001	1111	14
0010	0000	2
0010	0101	7
0010	1010	8
0010	1111	13
0011	0000	3
0011	0101	6
0011	1010	9
0011	1111	12

Figure 3.3. FX Distribution with I And IU_1 Transformation

Here, $IU_2(f_2) = \{0, 13\}$ and Device No = $T_M(I(J_1)\ [+]\ IU_2(J_2)), J_1 \in f_1, J_2 \in f_2$.

We can verify that the distribution of Figure 3.4 is perfect optimal.

3.4.4. U and IU_x Field Transformation Functions

We will show in Theorem 3.5 that for any values of F_j, F_k and M, FX distribution methods distribute ordered 2-tuples in $U(f_j) \times IU_x(f_k)$ optimally. Lemma 3.8 and Lemma 3.9 show useful properties of U and IU_x transformation functions, respectively. These two lemmas will be used to prove Theorem 3.5.

Definition 3.11. Let $S = \{s_1, \ldots, s_l\}$ be a set of nonnegative integers. Then for any nonnegative integer c we define $S + c = \{s_1 + c, \ldots, s_l + c\}$.

Lemma 3.8. Let $f_j = \{0, 1, \ldots, F_j - 1\}$ such that $F_j < M$. Let $d_j = M/F_j$. Then, for any nonnegative integer J and c such that $0 \le J < F_j$ and $0 \le c < d_j$, $U(f_j)\ [+]\ (Jd_j + c) = U(f_j) + c$.

$I(f_1)$	$IU_2(f_2)$	Device No
0000	0000	0
0000	1101	13
0001	0000	1
0001	1101	12
0010	0000	2
0010	1101	15
0011	0000	3
0011	1101	14
0100	0000	4
0100	1101	9
0101	0000	5
0101	1101	8
0110	0000	6
0110	1101	11
0111	0000	7
0111	1101	10

Figure 3.4. FX Distribution with I And IU_2 Transformation

Proof: By XOR uniqueness property, $|U(f_j)$ [+] $Jd_j| = F_j$. Let $\alpha d_j \in U(f_j)$. Then αd_j [+] Jd_j should be βd_j for some $\beta \in Z_{F_j}$. Thus, αd_j [+] $Jd_j \in U(f_j)$. Since $|U(f_j)$ [+] $Jd_j| = F_j$ and αd_j [+] $Jd_j \in U(f_j)$ for all $\alpha d_j \in U(f_j)$, $U(f_j)$ [+] Jd_j should be equal to $U(f_j)$. $Jd_j + c = Jd_j$ [+] c because $c < d_j$. Thus, $U(f_j)$ [+] Jd_j [+] $c = U(f_j)$ [+] $c = U(f_j) + c$. □

Lemma 3.9. Let $f_j = \{0, \ldots, F_j - 1\}$ such that $F_j^x < M$ for some positive integer x. Then, for any $K_1, K_2 \in f_j$, and for any d which is a power of 2, $K_1 \bmod d = K_2 \bmod d$ if and only if $IU_x(K_1) \bmod d = IU_x(K_2) \bmod d$.

Proof: The proof is an induction on x. Let $d_i = M/F_j^i$, $i = 1, \ldots, x$.

(1) Basis. $x = 1$. We want show that $K_1 \bmod d = K_2 \bmod d$ if and only if $(K_1$ [+] $K_1 d_1) \bmod d = (K_2$ [+] $K_2 d_1) \bmod d$. Let $\alpha = \log_2 d$ and $\beta = \log_2 d_1$.

Only if : When $d \le d_1$, bit positions 0 to $\alpha - 1$ of K_1 and K_2 are not affected by exclusive-or of $K_1 d_1$ and $K_2 d_1$. Therefore, this is true. When $d > d_1$, let $K_1 \bmod d =$

$K_2 \bmod d = (a_{\alpha-2} \cdots a_0)_B$. Then, $(K_1 [+] K_1 d_1) \bmod d = (a_{\alpha-2}[+]a_{\alpha-2-\beta}\ a_{\alpha-3}[+]a_{\alpha-3-\beta} \cdots a_0)_B$. Clearly, $(K_2 [+] K_2 d_1) \bmod d$ is also $(a_{\alpha-2}[+]a_{\alpha-2-\beta}\ a_{\alpha-3}[+]a_{\alpha-3-\beta} \cdots a_0)_B$.

If : Assume otherwise. Then there exist K_1, K_2 such that $(K_1 [+] K_1 d_1) \bmod d = (K_2 [+] K_2 d_1) \bmod d$, and $K_1 \bmod d \neq K_2 \bmod d$. Let $K_1 \bmod d = (a_{\alpha-2} \cdots a_0)_B$ and $K_2 \bmod d = (b_{\alpha-2} \cdots b_0)_B$. Let i be the rightmost bit position such that $a_i \neq b_i$. When $i < \beta$, clearly, contradiction. When $i \geq \beta$, $a_i [+] a_{i-\beta} = b_i [+] b_{i-\beta}$ because $(K_1 [+] K_1 d_1) \bmod d = (K_2 [+] K_2 d_1) \bmod d$ by the assumption. Since i is the rightmost bit position such that $a_i \neq b_i$, this implies that $a_{i-\beta} = b_{i-\beta}$. Therefore, a contradiction.

(2) Induction Step. Suppose this is true for $x = p$. We want to show that the lemma is also true for $x = p + 1$. (Note that we consider only the case where $F_j^{p+1} < M$). Let $(K_1 [+] K_1 d_1 [+] \ldots [+] K_1 d_p) = L_1$, and $(K_2 [+] K_2 d_1 [+] \ldots [+] K_2 d_p) = L_2$.

Only If : $K_1 d_{p+1} \bmod d$ is equal to $K_2 d_{p+1} \bmod d$. (It is easy to see that $K_1 \bmod d = K_2 \bmod d$ if and only if $K_1 d' \bmod d = K_2 d' \bmod d$ for any d' which is a power of 2.) Since $L_1 \bmod d = L_2 \bmod d$ by induction hypothesis, $(L_1 [+] K_1 d_{p+1}) \bmod d$ should be equal to $(L_2 [+] K_2 d_{p+1}) \bmod d$. Note that when d is a power of 2, $(L_1 [+] K_1 d_{p+1}) \bmod d = (L_1 \bmod d) [+] (K_1 d_{p+1} \bmod d)$.

If : Assume otherwise. Then there exist K_1, K_2 such that $(K_1[+]K_1 d_1[+] \cdots [+]K_1 d_{p+1}) \bmod d = (K_2[+]K_2 d_1[+] \cdots [+]K_2 d_{p+1}) \bmod d$, and $K_1 \bmod d \neq K_2 \bmod d$. Let $\alpha = \log_2 d$ and $\beta = \log_2 d_{p+1}$. Let $K_1 \bmod d = (a_{\alpha-2} \cdots a_0)_B$ and $K_2 \bmod d = (b_{\beta-2} \cdots b_0)_B$. Let i be the rightmost bit position such that $a_i \neq b_i$. The remainder of the proof is similar to that of case 1, i.e., $x = 1$. This completes the induction. □

Corollary 3.2. Let $f_j = \{0, \ldots, F_j - 1\}$ such that $F_j^x < M$ for some positive integer x. Let S_i, $i = 0, \ldots, d-1$ be a subset of $IU_x(f_j)$ whose residue by modulus d is i, where d is a power of 2 which is less than M. Then, for any such d, $|S_0| = |S_1| = \ldots =$

$|S_{d-1}|$.

Proof: This is a direct consequence of Lemma 3.9. □

Theorem 3.5. When there are only two fields j, k and the given number of devices is M such that $F_j < M$ and $F_k^x < M$ for some positive integer x, the FX distribution method with $U(f_j)$ and $IU_x(f_k)$ is perfect optimal.

Proof: Let $d_j = M/F_j$.

(case 1) $F_jF_k > M$ (i.e., $F_k > d_j$)

Let $F_jF_k = AM$. Then $A = F_k/d_j$, i.e., $F_k = Ad_j$. Let $K \in f_k$.

(1) K mod $d_j = 0$ (i.e., $K = Jd_j$ for some $J \in f_j$)

By Lemma 3.9, all $IU_x(K)$ elements such that K mod $d_j = 0$ has the same residue c_0 by modulus d_j. So, all such $IU_x(K)$ can be represented as $\alpha d_j + c_0$ for some variable $\alpha \in Z_{F_j}$ and some fixed nonnegative integer c_0 which is less than d_j. Since $F_k = Ad_j$, there are A number of $IU_x(K)$ such that K mod $d_j = 0$. For such $IU_x(K)$, $U(f_j)$ [+] $IU_x(K) = U(f_j) + c_0$ by Lemma 3.8. Let this set be S_0. Then there are A repetitions for each element in S_0 through $U(f_j)$ [+] $IU_x(K)$, $K = 0, d_j, \cdots, (A-1)d_j$.

(2) K mod $d_j = 1$ (i.e., $K = Jd_j + 1$, for some $J \in f_j$)

By Lemma 3.9 all $IU_x(K)$ elements such that K mod $d_j = 1$ has the same residue c_1 by modulus d_j. For all such K $U(f_j)$ [+] $IU_x(K) = U(f_j) + c_1$ by Lemma 3.8. Let this set be S_1. Since there are A number of $IU_x(K)$ such that K mod $d_j = 1$, there are A repetitions for each element in S_1 through $U(f_j)$ [+] $IU_x(K)$, $K = 1, d_j+1, \cdots, (A-1)d_j+1$.

-

-

(d_j-1) K mod $d_j = d_j - 1$

By Lemma 3.8 and Lemma 3.9, for any $IU_x(K)$ such that K mod $d_j = d_j-1$, $U(f_j)$ [+] $IU_x(K) = U(f_j) + c_{d_j-1}$ for some c_{d_j-1} which is less than d_j. Let this set be S_{d_j-1}. Then, there are A repetitions for each element in S_{d_j-1} through $U(f_j)$ [+] $IU_x(K)$, $K = d_j-1, \cdots, Ad_j-1$. By Lemma 3.9, all c_i's are different each other. So, $\bigcup_{l=0}^{d_j-1} S_l =$

Z_M. Since there are A repetitions for each element in Z_M through $U(f_j)$ [+] $IU_x(K)$, $K = 0, \cdots, F_k-1$, it is 2-optimal. 0 and 1-optimal come from Theorem 3.1.

(case 2) $F_jF_k < M$ (i.e., $F_k < d_j$)

The proof is almost the same as that of case 1. Here, enumeration ends at F_k-1 instead of d_j-1 and there exists only one $IU_x(K)$ element at each step. □

Example 3.7. Let $f_1 = \{0, 1, 2, 3\}$, $f_2 = \{0, 1, 2, 3\}$ and M = 16. Figure 3.5 shows the FX distribution with $U(f_1)$ and $IU_1(f_2)$. Here, $U(f_1) = \{0, 4, 8, 12\}$, $IU_1(f_2) = \{0, 5, 10, 15\}$ and Device No = $T_M(U(J_1)$ [+] $IU_1(J_2))$, $J_1 \in f_1, J_2 \in f_2$. We can see that the distribution of Figure 3.5 is perfect optimal.

$U(f_1)$	$IU_1(f_2)$	Device No
0000	0000	0
0000	0101	5
0000	1010	10
0000	1111	15
0100	0000	4
0100	0101	1
0100	1010	14
0100	1111	11
1000	0000	8
1000	0101	13
1000	1010	2
1000	1111	7
1100	0000	12
1100	0101	9
1100	1010	6
1100	1111	3

Figure 3.5. FX Distribution with U And IU_1 Transformation

Example 3.8. Let $f_1 = \{0, 1, 2, 3, 4, 5, 6, 7\}$, $f_2 = \{0, 1\}$ and M = 16. Figure 3.6 shows the FX distribution with $U(f_1)$ and $IU_2(f_2)$. Here, $U(f_1) = \{0, 2, 4, 6, 8, 10, 12, 14\}$, $IU_2(f_2) = \{0, 13\}$ and Device No = $T_M(U(J_1)$ [+] $IU_2(J_2))$, $J_1 \in f_1, J_2 \in f_2$. We can verify that the distribution of Figure 3.6 is perfect optimal.

$U(f_1)$	$IU_2(f_2)$	Device No
0000	0000	0
0000	1101	13
0010	0000	2
0010	1101	15
0100	0000	4
0100	1101	9
0110	0000	6
0110	1101	11
1000	0000	8
1000	1101	5
1010	0000	10
1010	1101	7
1100	0000	12
1100	1101	1
1110	0000	14
1110	1101	3

Figure 3.6. FX Distribution with U And IU_2 Transformation

3.4.5. IU_1 , IU_2 , . . . , IU_x Field Transformation Functions

In this section we will show that the set of IU_x transformation functions can give optimal distribution for the case when there are several fields whose sizes are much less than the given number of devices.

Definition 3.12. Let $<k_1, \ldots, k_n>$ and $<l_1, \ldots, l_n>$ be two ordered n-tuples. Then, $<k_1, \ldots, k_n> = <l_1, \ldots, l_n>$, if $k_1 = l_1, \ldots, k_n = l_n$. Otherwise, $<k_1, \ldots, k_n> \neq <l_1, \ldots, l_n>$.

Lemma 3.10. When there are only two fields $i1$, $i2$ such that $F_{i2} \geq F_{i1}$ and $F_{i2}{}^2 < M$ for the given number of devices M, the FX distribution method with $IU_1(f_{i1})$ and $IU_2(f_{i2})$ is perfect optimal.

Proof: Assume otherwise. Then, there exist $\{J_{11}, J_{12}\} \subseteq f_{i1}$, $\{J_{21}, J_{22}\} \subseteq f_{i2}$ such that $<J_{11}, J_{21}> \neq <J_{12}, J_{22}>$, and $(J_{11}[+]J_{11}d_{11})[+](J_{21}[+]J_{21}d_{21}[+]J_{21}d_{22}) = (J_{12}[+]J_{12}d_{11})[+](J_{22}[+]J_{22}d_{21}[+]J_{22}d_{22})$, where $d_{11} = M/F_{i1}$, $d_{21} = M/F_{i2}$, $d_{22} =$

d_{21}/F_{i2}. (Note that $F_{i1}F_{i2} < M$ and $d_{11} \geq d_{21}$.) The above equation can be rewritten as $[(J_{11}[+]J_{12})d_{11}[+](J_{21}[+]J_{22})d_{21}]$ [+] $[(J_{21}[+]J_{22})d_{22}[+]J_{11}[+]J_{12}[+]J_{21}[+]J_{22}]$ = 0. Since the second term (delimited by bracket) is less than d_{21}, and the first term does not have binary value "1" between bit position 0 and $\log_2 d_{21}$ - 1, the only way the above equation can be satisfied is that each term (delimited by bracket) should be equal to zero. Thus, we have following two equations.

$$(J_{11}[+]J_{12})d_{11}[+](J_{21}[+]J_{22})d_{21} = 0$$

$$(J_{21}[+]J_{22})d_{22}[+](J_{11}[+]J_{12}[+]J_{21}[+]J_{22}) = 0$$

(case 1) $d_{11} = d_{21}$

Then, $J_{11}[+]J_{12}[+]J_{21}[+]J_{22} = 0$ from the first equation. So, from the second equation J_{21} is equal to J_{22} which results in $J_{11} = J_{12}$. This contradicts $<J_{11},J_{21}> \neq <J_{21},J_{22}>$.

(case 2) $d_{11} > d_{21}$

Let $d_{11} = pd_{21}$, where p is a power of 2. From the first equation, $J_{21}[+]J_{22} = p(J_{11}[+]J_{12})$. Substituting this in the second equation, $(J_{11}[+]J_{12})pd_{22}[+](J_{11}[+]J_{12})[+](J_{11}[+]J_{12})p = 0$. Thus, the only way this equation can be satisfied is $J_{11} = J_{12}$ which implies $J_{21} = J_{22}$. Therefore, contradiction. □

Definition 3.13. We define LMB(J), J ∈ N, to be the leftmost bit position whose value is "1" in the binary notation of J, and define RMB(J), J ∈ N, to be the rightmost bit position whose value is "1" in the binary notation of J. When J = 0, we define LMB(J) = RMB(J) = -1.

For example, LMB(1) = 0, LMB(3) = 1, LMB(10) = 3, RMB(1) = 0, RMB(3) = 0, and RMB(10) = 1. Here, we considered the rightmost bit position to be the bit position zero.

Theorem 3.6. When there are only l fields, $i1$, $i2$, ... il such that $F_{il} \geq F_{i(l-1)} \geq ... \geq F_{i1}$ and $F_{il}{}^{l} < M$ for the given number of devices M, the FX distribution method with $IU_1(f_{i1}), ..., IU_l(f_{il})$ is perfect optimal.

Proof: The proof is an induction on l.

(1) When $l = 2$, this is true by Lemma 3.10.

(2) Suppose this is true until $l = k-1$. We want to show that this is true for $l = k$. For convenience, let these k fields be 1,2, ... , k. (Note that we only consider the case where $F_k \geq \ldots \geq F_1$, $F_k{}^k < M$, and so $F_1F_2\ldots F_k < M$.) Let $F_k = p_{k-1}F_{k-1} = p_{k-1}p_{k-2}F_{k-2} = \ldots = p_{k-1}p_{k-2}\cdots p_2F_2 = p_{k-1}p_{k-2}\cdots p_2p_1F_1$, where each p_i, $i = 1, \ldots$, k-1, is either 1 or some power of 2. Let $d_{11} = M/F_1$, and $d_{21} = M/F_2$, $d_{22} = M/F_2{}^2$, and . . . , and $d_{k1} = M/F_k, \ldots, d_{kk} = M/F_k{}^k$. Then,

$$d_{11} = p_1d_{21} = p_1p_2d_{31} = p_1p_2p_3d_{41} = \ldots = p_1p_2\cdots p_{k-2}d_{(k-1)1} = p_1p_2\cdots p_{k-1}d_{k1}$$

$$d_{22} = p_2{}^2d_{32} = p_2{}^2p_3{}^2d_{42} = \ldots = p_2{}^2p_3{}^2\ldots p_{k-2}{}^2d_{(k-1)2} = p_2{}^2p_3{}^2\ldots p_{k-1}{}^2d_{k2}$$

$$d_{33} = p_3{}^3d_{43} = p_3{}^3p_4{}^3d_{53} = \ldots = p_3{}^3p_4{}^3\ldots p_{k-2}{}^3d_{(k-1)3} = p_3{}^3p_4{}^3\ldots p_{k-1}{}^3d_{k3}$$

-

-

$$d_{(k-2)(k-2)} = p_{k-2}{}^{k-2}d_{(k-1)(k-2)} = p_{k-2}{}^{k-2}p_{k-1}{}^{k-2}d_{k(k-2)}$$

$$d_{(k-1)(k-1)} = p_{k-1}{}^{k-1}d_{k(k-1)}$$

Assume otherwise, i.e., there exist $\{J_{11},J_{12}\} \subseteq f_1, \ldots, \{J_{k1},J_{k2}\} \subseteq f_k$ such that $\langle J_{11}, \ldots, J_{k1}\rangle \neq \langle J_{12}, \ldots, J_{k2}\rangle$ and

$$(J_{11}[+]J_{11}d_{11})[+](J_{21}[+]J_{21}d_{21}[+]J_{21}d_{22})[+] \ . \ . \ . \ [+]\ (J_{k1}d_{k1}[+]\ldots[+]J_{k1}d_{kk}) =$$

$$(J_{12}[+]J_{12}d_{11})[+](J_{22}[+]J_{22}d_{21}[+]J_{22}d_{22})[+] \ . \ . \ . \ [+]\ (J_{k2}d_{k2}[+]\ldots[+]J_{k2}d_{kk})$$

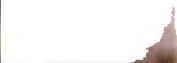
Here, $J_{k1} \neq J_{k2}$. This is because if $J_{k1} = J_{k2}$, then $\langle J_{11}, \ldots, J_{(k-1)1}\rangle \neq \langle J_{12}, \ldots, J_{(k-1)2}\rangle$ and hence the equality cannot be satisfied by the induction hypothesis and XOR uniqueness property. The above equality can be rewritten as

$$[(J_{11}[+]J_{12})d_{11}[+](J_{21}[+]J_{22})d_{21}\ [+] \ . \ . \ . \ [+]\ (J_{(k-1)1}[+]J_{(k-1)2})d_{(k-1)1}[+](J_{k1}[+]J_{k2})d_{k1}]\ [+]$$

$$[(J_{21}\ [+]\ J_{22})d_{22}\ [+] \ . \ . \ . \ [+]\ (J_{(k-1)1}\ [+]\ J_{(k-1)2})d_{(k-1)2}\ [+]\ (J_{k1}\ [+]\ J_{k2})d_{k2}]\ [+]$$

-

-



$$[(J_{(k-2)1} [+] J_{(k-2)2})d_{(k-2)(k-2)} [+] (J_{(k-1)1} [+] J_{(k-1)2})d_{(k-1)(k-2)} [+] (J_{k1} [+] J_{k2})d_{k(k-2)}] [+]$$

$$[(J_{(k-1)1} [+] J_{(k-1)2})d_{(k-1)(k-1)} [+] (J_{k1} [+] J_{k2})d_{k(k-1)}] [+]$$

$$[(J_{k1} [+] J_{k2})d_{kk} [+] (J_{11} [+] J_{12}) [+] (J_{21} [+] J_{22}) [+] \ldots [+] (J_{k1} [+] J_{k2})] = 0.$$

Now, we will show that each term delimited by bracket (i.e., each line in the above equation) should be equal to zero.

(1) The k-th term (i.e., the last term) should be equal to zero because the last term cannot be greater than $d_{k(k-1)}$, and all other terms do not have binary value "1" between bit position 0 and $\log_2 d_{k(k-1)} - 1$,

(2) Suppose (k-1)-th term is not zero. Then, $(J_{(k-1)1} [+] J_{(k-1)2})d_{(k-1)(k-1)} \geq d_{k(k-2)}$. This is because all other terms do not have binary value "1" between bit position $\log_2 d_{k(k-1)}$ and $\log_2 d_{k(k-2)} - 1$. So, the only way this inequality can be satisfied is $LMB((J_{(k-1)1} [+] J_{(k-1)2})d_{(k-1)(k-1)}) \geq RMB((J_{k1} [+] J_{k2})d_{k(k-2)})$. (If not, we can see immediately either (k-1)-th term should be zero or there is no way the whole equation can be satisfied.) Since $RMB((J_{(k-1)1} [+] J_{(k-1)2})d_{(k-1)(k-1)})$ should be equal to $RMB((J_{k1} [+] J_{k2})d_{k(k-1)})$, and $RMB((J_{k1} [+] J_{k2})d_{k(k-2)}) - RMB((J_{k1} [+] J_{k2})d_{k(k-1)}) = \log_2 F_k$, it follows that $(J_{(k-1)1} [+] J_{(k-1)2}) > F_k$. This contradicts $F_{k-1} \leq F_k$. Hence (k-1)-th term should be equal to zero.

(3) Suppose (k-2)-th term is not zero. Then, $(J_{(k-2)1} [+] J_{(k-2)2})d_{(k-2)(k-2)}$ should be greater than or equal to $d_{k(k-3)}$. This is because first, all the other terms do not have binary value "1" between bit position $\log_2 d_{k(k-2)}$ and $\log_2 d_{k(k-3)} - 1$, and second, from (2) $(J_{(k-1)1} [+] J_{(k-1)2})d_{(k-1)(k-1)} < d_{k(k-2)}$ and this implies that $(J_{(k-1)1} [+] J_{(k-1)2})d_{(k-1)(k-2)}$ is less than $d_{k(k-3)}$. (If we multiply F_{k-1} to the left-hand-side, and multiply F_k to the right-hand-side of the inequality $(J_{(k-1)1} [+] J_{(k-1)2})d_{(k-1)(k-1)} < d_{k(k-2)}$, this statement follows). The remainder of the proof is almost the same as that of case (2)

By continuing this argument, all the terms delimited by bracket should be equal to zero. Thus, we have the following k equalities.

$$(J_{11}[+]J_{12})p_1 \cdots p_{k-1}\ [+]\ (J_{21}[+]J_{22})p_2 \cdots p_{k-1}\ [+]\ \ldots\ [+]\ (J_{(k-1)1}[+]J_{(k-1)2})p_{k-1}\ [+]\ (J_{k1}[+]J_{k2}) = 0$$

$$(J_{21}[+]J_{22}){p_2}^2 \cdots {p_{k-1}}^2\ [+]\ \ldots\ [+]\ (J_{(k-1)1}[+]J_{(k-1)2})p_{k-1}{}^2\ [+]\ (J_{k1}[+]J_{k2}) = 0$$

$$\vdots$$

$$(J_{(k-2)1}[+]J_{(k-2)2}){p_{k-2}}^{k-2}p_{k-1}{}^{k-2}\ [+]\ (J_{(k-1)1}[+]J_{(k-1)2})p_{k-1}{}^{k-2}\ [+]\ (J_{k1}[+]J_{k2}) = 0$$

$$(J_{(k-1)1}[+]J_{(k-1)2})p_{k-1}{}^{k-1}\ [+]\ (J_{k1}[+]J_{k2}) = 0$$

$$(J_{k1}[+]J_{k2})d_{kk}\ [+]\ (J_{11}[+]J_{12})\ [+]\ \ldots\ [+]\ (J_{k1}[+]J_{k2}) = 0$$

Now, assume $p_{k-1} = 1$. Then, from (k-1)-th equality $(J_{(k-1)1}[+]J_{(k-1)2})\ [+]\ (J_{k1}[+]J_{k2}) = 0$, and so from the (k-2)-th equality $J_{(k-2)1} = J_{(k-2)2}$. This results in $J_{(k-3)1} = J_{(k-3)2}$ from (k-3)-th equality, and results in $J_{(k-4)1} = J_{(k-4)2}, \ldots, J_{11} = J_{12}$. So, from the last equality $J_{k1} = J_{k2}$, and hence $J_{(k-1)1} = J_{(k-1)2}$. This contradicts $<J_{11}, \ldots, J_{k1}> \neq <J_{12}, \ldots, J_{k2}>$. Thus, p_{k-1} should be greater than 1.

In the (k-1)-th equality, since p_{k-1} is a power of 2, $LMB\,(J_{k1}[+]J_{k2})$ should be greater than $LMB\,(J_{(k-1)1}[+]J_{(k-1)2})$. (Otherwise, (k-1)-th equality cannot be satisfied.) By the similar reason, it is easy to see that $LMB\,(J_{k1}[+]J_{k2}) > LMB\,(J_{i1}[+]J_{i2})$, $i = 1, \ldots$ k-1. Now, from the last equality, since $LMB\,(J_{k1}[+]J_{k2}) > LMB\,(J_{i1}[+]J_{i2})$, $i = 1, \ldots$ k-1 and d_{kk} is a power of 2, there is no way the last equality can be satisfied. We have shown that there do not exist $\{J_{11}, J_{12}\} \subseteq f_1, \ldots, \{J_{k1}, J_{k2}\} \subseteq f_k$ such that $<J_{11}, \ldots, J_{k1}> \neq <J_{12}, \ldots, J_{k2}>$ and $IU_1(J_{11})\ [+]\ \ldots\ [+]\ IU_k(J_{k1}) = IU_1(J_{12})\ [+]\ \ldots\ [+]\ IU_k(J_{k2})$. This completes the proof for $l = k$. Thus the theorem follows by the principle of induction. □

The following example shows the proof of Theorem 3.6 when $l = 3$, i.e., for IU_1, IU_2 and IU_3.

Example 3.9. Let a file consist of three fields 1, 2, 3 such that $F_3 \geq F_2 \geq F_1$ and ${F_3}^3 <$ M for the given number of devices M. Suppose FX distribution with $IU_1(f_1)$, $IU_2(f_2)$ and $IU_3(f_3)$ is not perfect optimal. Then, there exist $\{J_{11}, J_{12}\} \subseteq f_1$, $\{J_{21}, J_{22}\} \subseteq f_2$, $\{J_{31}, J_{32}\} \subseteq f_3$ such that $<J_{11}, J_{21}, J_{31}> \neq <J_{12}, J_{22}, J_{32}>$ and

$(J_{11}[+]J_{11}d_{11})[+](J_{21}[+]J_{21}d_{21}[+]J_{21}d_{22})[+](J_{31}[+]J_{31}d_{31}[+]J_{31}d_{32}[+]J_{31}d_{33}) =$

$(J_{12}[+]J_{12}d_{11})[+](J_{22}[+]J_{22}d_{21}[+]J_{22}d_{22})[+](J_{32}[+]J_{32}d_{31}[+]J_{32}d_{32}[+]J_{32}d_{33})$

where $d_{11} = M/F_1$, $d_{21} = M/F_2$, $d_{22} = d_{21}/F_2$, $d_{31} = M/F_3$, $d_{32} = d_{31}/F_3$, $d_{33} = d_{32}/F_3$. Here, $J_{31} \neq J_{32}$. This is because if $J_{31} = J_{32}$, then $<J_{11}, J_{21}> \neq <J_{12}, J_{22}>$ and hence the equality cannot be satisfied by Lemma 3.10 and XOR uniqueness property. The above equality can be rewritten as,

$$[(J_{11}[+]J_{12})d_{11}\ [+]\ (J_{21}[+]J_{22})d_{21}\ [+]\ (J_{31}[+]J_{32})d_{31}]\ [+]$$

$$[(J_{21}[+]J_{22})d_{22}\ [+]\ (J_{31}[+]J_{32})d_{32}]\ [+]$$

$$[(J_{31}[+]J_{32})d_{33}\ [+]\ (J_{11}[+]J_{12})\ [+]\ (J_{21}[+]J_{22})\ [+]\ (J_{31}[+]J_{32})] = 0$$

Let $F_3 = p_2F_2 = p_1p_2F_1$, where each p_i, $i = 1,2$ is either one or some power of 2. Then, $d_{11} = p_1d_{21} = p_1p_2d_{31}$, and $d_{22} = {p_2}^2d_{32}$.

Now, we will show that each term delimited by bracket (i.e., each line) should be equal to zero.

(1) The third term (i.e., the last term) should be equal to zero because the last term is less than d_{32}, and the first and the second term do not have any binary value "1" between bit position 0 and $\log_2 d_{32} - 1$.

(2) Suppose the second term is not equal to zero. Then, $(J_{21}[+]J_{22})d_{22} > d_{31}$ because the first term does not have binary value "1" between bit position $\log_2 d_{32}$ and $\log_2 d_{31} - 1$. Hence, $LMB((J_{21}[+]J_{22})d_{22}) \geq RMB((J_{31}[+]J_{32})d_{31})$. Otherwise, there is no way the equality can be satisfied. (Here, note that $(J_{21}[+]J_{22})d_{22} < d_{21} \leq d_{11}$). Since $RMB((J_{21}[+]J_{22})d_{22})$ should be equal to $RMB((J_{31}[+]J_{32})d_{32})$, and

$RMB((J_{31}[+]J_{32})d_{31})$ - $RMB((J_{31}[+]J_{32})d_{32}) = \log_2 F_3$, it follows that $(J_{21}[+]J_{22}) > F_3$. This contradicts $F_2 \leq F_3$. Thus, the second term should be equal to zero, and hence the first term is also zero. Now, we have the following three equalities.

$$(J_{11}[+]J_{12})p_1p_2d_{31}\ [+]\ (J_{21}[+]J_{22})p_2d_{31}\ [+]\ (J_{31}[+]J_{32})d_{31} = 0$$

$$(J_{21}[+]J_{22})p_2{}^2d_{32}\ [+]\ (J_{31}[+]J_{32})d_{32} = 0$$

$$(J_{31}[+]J_{32})d_{33}\ [+]\ (J_{11}[+]J_{12})\ [+]\ (J_{21}[+]J_{22})\ [+]\ (J_{31}[+]J_{32}) = 0$$

Here, assume $p_2 = 1$. Then from the second equality, $(J_{21}[+]J_{22})\ [+]\ (J_{31}[+]J_{32}) = 0$, and so from the first equality $J_{11} = J_{12}$ which results in $J_{31} = J_{32}$ in the last equality. This contradicts $J_{31} \neq J_{32}$. Thus, P_2 should be greater than one.

In the second equality, since p_2 is a power of 2, $LMB(J_{31}[+]J_{32})$ should be greater than $LMB(J_{21}[+]J_{22})$. Thus, from the first equality, it is easy to see that $LMB(J_{31}[+]J_{32}) > LMB(J_{11}[+]J_{12})$. Then, there is no way the third equality can be satisfied except $J_{31} = J_{32}$ which is contradiction. Therefore, the FX distribution method with $IU_1(f_1)$, $IU_2(f_2)$ and $IU_3(f_3)$ is perfect optimal.

Corollary 3.3. Let a file consist of l fields, 1, 2, , . . ,l such that $F_l \geq F_{l-1} \geq \ldots \geq F_1$ and $F_l{}^l$ < M for the given number of devices M. Then, for any set of fields $\{i1, \ldots, ik\} \subseteq \{1, 2, , . . ,l\}$, the FX distribution method with $IU_{i1}(f_{i1}), \ \ldots\ , IU_{ik}(f_{ik})$ is perfect optimal.

Proof: This is a direct consequence of Theorem 3.6. □

3.4.6. I, U and IU_x Field Transformation Functions

In this section we show that FX distribution methods can always give perfect optimal distribution as long as the number of fields, whose sizes are less than the given number of devices, is no greater than 4. Lemma 3.11 and Lemma 3.12 shows useful properties of IU_x transformation functions in proving Theorem 3.7.

Lemma 3.11. Let $f_k = \{0, \ldots, F_k\text{-}1\}$ such that $F_k{}^x < M$ for some positive integer x. Let $d_i = M/F_k{}^i$, $i = 1, \ldots, x$. Let $K_1, K_2 \in f_k$, and α be some power of 2 which is less than F_k. Then, K_1 and K_2 are in the same interval of size α from 0 to F_k if and only if $IU_x(K_1)$ and $IU_x(K_2)$ are in the same interval of size αd_1 from 0 to M.

Proof:

Only If : When K_1 and K_2 are in the same interval of size α, $(K_1 [+] K_2) \leq \alpha\text{-}1$ and hence $(K_1 [+] K_2)d_1 \leq (\alpha\text{-}1)d_1$. Now,

$IU_x(K_1) [+] IU_x(K_2)$

$= (K_1 [+] K_1d_1 [+] \ldots [+] K_1d_x) [+] (K_2 [+] K_2d_1 [+] \ldots [+] K_2d_x)$

$= (K_1[+]K_2)d_1[+][(K_1[+]K_1d_2[+] \ldots [+]K_1d_x)[+](K_2[+]K_2d_2[+] \ldots [+]K_2d_x)]$

$< (K_1 [+] K_2)d_1 + d_1$

The last inequality holds because $[(K_1 [+]K_1d_2 [+] \ldots [+] K_1d_x) [+] (K_2 [+]K_2d_2 [+] \ldots [+] K_2d_x)] < d_1$ by the construction of d_i, $i = 1, \ldots, x$. Thus, $IU_x(K_1) [+] IU_x(K_2) < \alpha d_1$. This implies that $IU_x(K_1)$ and $IU_x(K_2)$ are in the same interval of size αd_1.

If : When $IU_x(K_1)$ and $IU_x(K_2)$ are in the same interval of size αd_1, $(K_1 [+] K_1d_1 [+] \ldots [+] K_1d_x) [+] (K_2 [+] K_2d_1 [+] \ldots [+] K_2d_x) < \alpha d_1$. So, $(K_1 [+] K_2)d_1 + [(K_1[+]K_1d_2[+] \ldots [+]K_1d_x)[+] (K_2[+]K_2d_2[+] \ldots [+]K_2d_x)] < \alpha d_1$. Thus, $(K_1 [+] K_2)d_1 < \alpha d_1$, and so $K_1 [+] K_2 < \alpha$. □

Lemma 3.12. Let $f_k = \{0, \ldots, F_k\text{-}1\}$ such that $F_k{}^x < M$ for some positive integer x. Let $d_i = M/F_k{}^i$, $i = 1, \ldots x$. Let $w = \alpha d_1$ in which α is a power of 2, and let β be a power of 2 which is less than or equal to α. For some nonnegative integer c which is less than M/w, let $S_i^{(cw)} = \{K \in f_k \mid cw \leq IU_x(K) < (c+1)w, K \bmod \beta = i\}$. Then for any such w, α, β, c, $|S_0{}^{(cw)}| = |S_1{}^{(cw)}| = \ldots |S_\beta{}^{(cw)}|$.

Proof: By Lemma 3.7, for any nonnegative integer c which is less than M/w, $\#\{s \in IU_x(f_k) \mid cw \leq s < (c+1)w\} = \alpha$. Let $\{K_1, \ldots K_\alpha\}$ be the subset of f_k such that

$cw \le IU_x(K_i) < (c+1)w$, $i=1,...,\alpha$. Then by Lemma 3.11, all $K_1, \ldots, K_\alpha$ are in the same interval with size α, and hence $\{K_1, \ldots, K_\alpha\}$ is a set of α consecutive integers. Thus the lemma follows. □

Theorem 3.7. When there are only three fields i, j, k whose sizes are less than the given number of devices M, and $F_k^x < M$ for some positive integer x which is greater than 1, the FX distribution methods with $I(f_i)$, $U(f_j)$ and $IU_x(f_k)$ is perfect optimal, if either (i) there are at least 2 fields r and s such that $r, s \in \{i, j, k\}$ and $F_rF_s \ge M$ or (ii) $F_k \ge F_j$

Proof: It is clear that (i) is a sufficient condition for perfect optimal distribution by Theorem 3.1, 3.3, 3.4, 3.5 and Proposition 3.1. So, let us consider only the other case, i.e., $F_rF_s < M$ for any $r, s \in \{i, j, k\}$. We want to show that (ii) is sufficient for perfect optimal distribution for this case. Let $d_j = M/F_j$ and $d_{kl} = M/F_k^l$, $l = 1, \cdots, x$. Then, $d_j > F_i$, $d_j > F_k$ and $d_j > d_{k1}$ and $d_{k1} > F_i$, $d_{k1} > F_j$.

(case 1) $F_iF_jF_k \ge M$

Let $F_iF_jF_k = AM$, and let $d_j = Bd_{k1}$ and $d_{k1} = CF_i$, where A,B and C are some positive integers. Then, $F_iM/d_jF_k = AM$ or $F_i = Ad_j/F_k = ABd_{k2}$. Since $d_j > F_i$, $I(f_i)$ [+] $U(f_j) = S_0 \cup S_1 \cup ... \cup S_{F_j-1}$, where

$$S_0 = \{0, 1, ..., F_i-1\}$$
$$S_1 = \{d_j, d_j+1, \ldots, d_j+F_i-1\}$$
$$\ldots$$
$$S_i = \{id_j, id_j+1, \ldots, (i+1)d_j+F_i-1\}$$
$$\ldots$$
$$S_{F_j-1} = \{(F_j-1)d_j, (F_j-1)d_j+1, \ldots, (F_j-1)d_j+F_i-1\}$$

Clearly, all S_i's are disjoint. By definition 3.11 $S_i+c = \{id_j+c, id_j+1+c, \ldots, id_j+F_i-1+c\}$ for any positive integer c which is less than d_j. Let $K \in f_k$. By Lemma 3.7 there is exactly one element of $K[+]Kd_{k2}[+] \ldots [+]Kd_{kx}$ at each interval from 0 to d_{k1} with interval size d_{k2} (Note that this can be obtained by substituting d_{k1} for M in

Lemma 3.7).

(1) $0 \le (K\ [+]\ K\ d_{k2}\ [+] \ldots [+]\ Kd_{kx}) < F_i$

Since F_i = ABd_{k2}, there are AB number of $IU_x(K)$ such that $0 \le (K\ [+]\ K\ d_{k2}\ [+] \ldots [+]\ Kd_{kx}) < F_i$.

(1-1) K mod B = 0 (i.e., $Kd_{k1} \bmod d_j = 0$)

For each $IU_x(K)$ within this range, $I(f_i)\ [+]\ U(f_j)\ [+]\ IU_x(K) = \bigcup_{i=0}^{F_j-1} S_i$ The equality holds due to Lemma 3.1, and Lemma 3.2 (Note that $K[+]Kd_{k1}[+] \cdots [+]Kd_{k2} = Kd_{k1}[+](K\ [+]\ Kd_{k2}[+]...[+]Kd_{kx})$). Let this set be T_{11}. Since there are A number of such $IU_x(K)$'s within this range by Lemma 3.12 (this statement follows by substitute d_{k1} for M in Lemma 3.12), there are A repetitions for each element in T_{11}.

(1-2) K mod B = 1 (i.e., $Kd_{k1} \bmod d_j = d_{k1}$)

For each $IU_x(K)$ within this range, $I(f_i)\ [+]\ U(f_j)\ [+]\ IU_x(K) = \bigcup_{i=0}^{F_j-1} S_i + d_{k1}$. The equality holds due to Lemma 3.1 and Lemma 3.2. (Note that $d_{k1} < d_j$.) Let this set be T_{12}. Since there are A number of such $IU_x(K)$ elements within this range by Lemma 3.12, there are A repetitions for each element in T_{12}

-

-

(1-B) K mod B = B − 1 (i.e., $Kd_{k1} \bmod d_j = (B-1)d_{k1}$)

For each $IU_x(K)$ within this range, $I(f_i)\ [+]\ U(f_j)\ [+]\ IU_x(K) = \bigcup_{i=0}^{F_j-1} S_i + (B-1)d_{k1}$. Let this set be T_{1B}. Since there are A umber of such $IU_x(K)$ elements within this range, there are A repetitions for each element in T_{1B}.

(2) $F_i \le K\ [+]\ Kd_{k2} < 2F_i$

(2-1) K mod B = 0

For each $IU_x(K)$ within this range, $I(f_i)\ [+]\ U(f_j)\ [+]\ IU_x(K) = \bigcup_{i=0}^{F_j-1} S_i + F_i$. Let this set

be

T_{21}. By the same reason as previous, there are A repetitions for each element in T_{21} .

-

-

(2-B) K mod B = B − 1

For each $IU_x(K)$ within this range, $I(f_i)$ [+] $U(f_j)$ [+] $IU_x(K) = \bigcup_{i=0}^{F_j-1} S_i + (B-1)d_{k1} + F_i$. (Note that $d_{k1} > F_i$.) Let this set be T_{2B}. By the same reason as previous, there are A repetitions for each element in T_{2B}.

-

-

(C) $(C-1)F_i \le K$ [+] $Kd_{k2} < CF_i$

(C-1) K mod B = 0

For each $IU_x(K)$ within this range, $I(f_i)$ [+] $U(f_j)$ [+] $IU_x(K) = \bigcup_{i=0}^{F_j-1} S_i + (C-1)F_i$. (Note that $CF_i = d_{k1}$.) Let this set be T_{C1}. By the same reason as previous, there are A repetitions for each element in T_{C1}.

-

-

(C-B) K mod B = B - 1

For each $IU_x(K)$ within this range, $I(f_i)$ [+] $U(f_j)$ [+] $IU_x(K) = \bigcup_{i=0}^{F_j-1} S_i + (B-1)d_{k1} + (C-1)F_i$. Let this set be T_{CB}. By the same reason as previous, there are A repetitions for each element in T_{CB}. Now, it is not difficult to see that $\bigcup_{i=1,j=1}^{i=C,j=B} T_{ij} = Z_M$ and all T_{ij}'s are disjoint. Since we already show that there are A repetitions for every element in $\bigcup_{i=1,j=1}^{i=C,j=B} T_{ij}$, it is 3-optimal. 2-optimal come from Theorem

3.3, 7, 8. 0 and 1-optimal come from Theorem 3.1.

(case 2) $F_iF_jF_k < M$

The proof is almost the same as that of case 1. □

Corollary 3.4. Let L be the number of fields whose sizes are less than the given number of devices M. FX distribution methods can be always perfect optimal, if $L \leq 3$.

Proof: When L = 0, 1, 2, it follows from Theorem 3.1, 3.2, 3.3, 3.4, 3.5 and Proposition 3.1. When L = 3, let i, j, k are fields whose sizes are less than M and $F_i \geq F_k \geq F_j$. If $F_k^2 \geq M$, apply $I(f_i)$, $U(f_j)$ and $IU_1(f_k)$ transformation. Since $F_iF_k \geq M$, the corollary follows by (i) of Theorem 3.7. If $F_k^2 < M$, apply $I(f_i)$, $U(f_j)$ and $IU_2(f_k)$ transformation. Then the corollary follows by (ii) of Theorem 3.7. □

Example 3.10. Let $f_1 = \{0, 1, 2, 3\}$, $f_2 = \{0, 1\}$, $f_3 = \{0, 1\}$ and M = 8. Figure 3.7 shows the FX distribution with $I(f_1)$, $U(f_2)$ and $IU_2(f_3)$. Here, $U(f_2) = \{0, 4\}$ and $IU_2(f_3) = \{0, 7\}$, and Device No = $T_M(I(J_1)\ [+]\ U(J_2)\ [+]\ IU_2(J_3))$, $J_1 \in f_1, J_2 \in f_2$, $J_3 \in f_3$. (Here, we can also verify that the FX distribution with $I(f_1)$, $U(f_2)$ and $IU_1(f_3)$ gives perfect optimal distribution because the file system of this example also satisfies condition (i) of Theorem 3.7).

We have shown that by various combinations of field transformation functions FX distribution methods give strict optimal distribution for many types of partial match queries. Here, it should be emphasized that these field transformation techniques along with Proposition 3.1 increase the scope of optimality considerably. This is because by Proposition 3.1 optimal distribution for a subset of fields guarantees strict optimal distribution for many partial match queries in which those fields are unspecified. For example, let a partial match query unspecify fields 1, 2, 3, where $|f_1| = 8$, $|f_2| = 4$, $|f_3| = 8$, and M = 32. When field 1 is U transformed and field 2 is I-transformed, then regardless of field 3, the distribution is strict optimal for this query by Theorem 3.3 and Proposition 3.1.

$I(f_1)$	$U(f_2)$	$IU_2(f_3)$	Device No
000	000	000	0
000	000	111	7
000	100	000	4
000	100	111	3
001	000	000	1
001	000	111	6
001	100	000	5
001	100	111	2
010	000	000	2
010	000	111	5
010	100	000	6
010	100	111	1
011	000	000	3
011	000	111	4
011	100	000	7
011	100	111	0

Figure 3.7. FX Distribution with I, U And IU_2 Transformation

Now, the summarized results of FX distribution methods are as follows (here, all the theorems, lemmas and corollary in section 3.3 also hold for (Extended) FX distribution methods) :

Let a file consist of n fields and there be M parallel devices. Let L be the number of fields whose sizes are less than the given number of devices M. FX distribution methods can be always perfect optimal, when $L \leq 3$. Let L be greater than or equal to 4. Let $q_u(f)$ be the set of fields which are unspecified for partial match query q. Then, FX distribution methods are strict optimal for partial match query q, if at least one of the following conditions holds.

(1) $| q_u(f) | = 0$ or 1

(2) there is at least one field $i \in q_u(f)$ such that $F_i \geq M$.

(3) $| q_u(f) | = 2$ and transformation methods of two fields in $q_u(f)$ are different. (ref. section 3.4.1.)

(4) $| q_u(f) | = 3$ and either

(a) there are at least two fields i, $j \in q_u(f)$ such that $F_iF_j \geq M$ and transformation methods of two fields i and j are different, or

(b) transformation methods of three fields in $q_u(f)$ are I, U, IU_x for some $x \geq 2$, and the size of IU_x transformed field is not less than the size of U transformed field.

(5) $| q_u(f) | \geq 4$ and either

(a) there are at least two fields i, $j \in q_u(f)$ such that $F_iF_j \geq M$ and transformation methods of two fields i and j are different, or

(b) there are at least three fields i, j, k $\in q_u(f)$ such that $F_iF_jF_k \geq M$ and transformation methods of fields i, j and k are I, U, IU_x for some $x \geq 2$, and the size of IU_x transformed field is not less than the size of U transformed field.

(c) the fields in $q_u(f)$ satisfy Theorem 3.6

Note that these are only sufficient and are not necessary conditions.

It is unfortunate that FX distribution methods do not always guarantee perfect optimal distribution when the number of fields whose sizes are less than M, is greater than or equal to 4 in general. In fact, it has been shown in [Sun87] that when the number of fields whose sizes are less than the given number of devices, is greater than or equal to 4, there is no method which always gives perfect optimal distribution. However, even for these cases we will show through performance experiments that FX distribution methods still gives near optimal distribution for most queries.

3.5. Performance Comparison with Other Distribution Methods

In this section we compare the performance of FX distribution methods with those of the DM and GDM distribution method. The performance comparisons are based on the probability of strict optimality and the average response time for a given partial

match query. In section 3.5.1 the probability of strict optimal distribution for partial match queries is given. In section 3.5.2 we compare the average response time for the FX, DM and GDM distribution method.

For both section 3.5.1 and 3.5.2, it is assumed that the probability of each field being specified is same for all fields and some field being specified is independent of each other.

3.5.1. Probability of Strict Optimality

In this section we show that the probability of strict optimality for FX distribution methods is much higher than the DM distribution method. Even for the worst case the decrease of probability of strict optimality for FX distribution is not much. On the other hand, in the DM distribution the decrease is quite large. Since no general method has been given to determine the existence of parameter values for strict optimal distribution in the GDM method, we compare FX distribution methods to DM distribution method only.

Let the file consist of 7 fields and M = 32. Let $F_1 = 2$, $F_2 = F_3 = 4$, $F_4 = F_5 = F_6 = 8$, $F_7 = 16$. In the DM distribution method the probability of strict optimal distribution for some partial match query is 0.0547 (computed from the optimality conditions given in [DuS82]). On the other hand, in FX distribution methods with $IU_1(f_1)$, $IU_2(f_2)$, $U(f_3)$, $I(f_4)$, $U(f_5)$, $IU_1(f_6)$, $I(f_7)$, the probability of strict optimal for a partial match query is 0.9531 (computed from the sufficient conditions in section 3.3 and 3.4). Therefore, in this example FX distribution methods give much higher probability of strict optimality than the DM distribution method.

Figure 3.8 and 3.9 show the percentage of strict optimal distribution for all possible partial match queries in a given file system. In these figures DM denotes the results of the DM distribution method and FX denotes the results of FX distribution methods. Here, the results are computed from sufficient conditions given for each method. Figure

3.8 shows the case where for any two fields r and s whose sizes are less than the given number of devices, $F_rF_s \geq M$. We used files with six and ten fields. In this figure FX distribution methods used I, U and IU_1 transformation methods.

Figure 3.9 shows the percentage of strict optimal distribution when for any two fields r and s whose sizes are less than the given number of devices, $F_rF_s < M$, but for any three fields r, s and t whose sizes are less than the given number of devices, $F_rF_sF_t \geq M$. Here, in FX distribution methods I, U and IU_2 transformation methods are used.

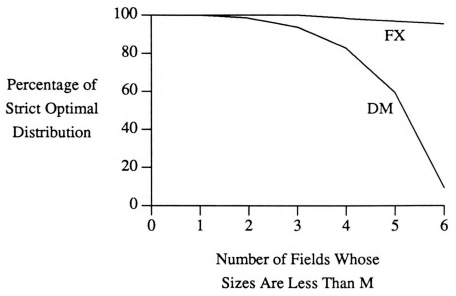
These results show that FX distribution methods give high probability of strict optimal distribution for partial match queries in typical file systems. However, since there is no convenient way to compute probability of strict optimal distribution in more general file systems, in the next section we give results of performance experiments based on average response time.

3.5.2. Average Response Time

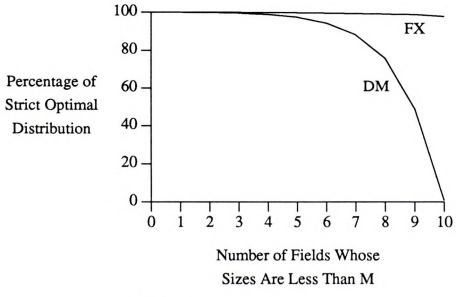
Definition 3.14. For a given partial match query q, $r_i(q)$ is defined as the number of qualified buckets in device i for a partial match query q. We call this the *response size* of device i for a partial match query q. Then, the *largest response size* for a partial match query q is defined as $MAX(r_1(q), r_2(q), \cdots r_{M-1}(q))$.

For the response time of a partial match query, we will consider two factors, namely, largest response size and CPU computation time for bucket address calculation. In parallel disks environment, largest response size is the most important factor, while in main memory databases, CPU computation time is more important.

When systems are configured such that the data retrieval time for any device is almost the same, the response time for a partial match query is determined by the device which has the largest number of qualified buckets. For example, parallel disks connected to one shared bus, or some of the multiprocessors based on multistage interconnection networks are considered to be such systems.

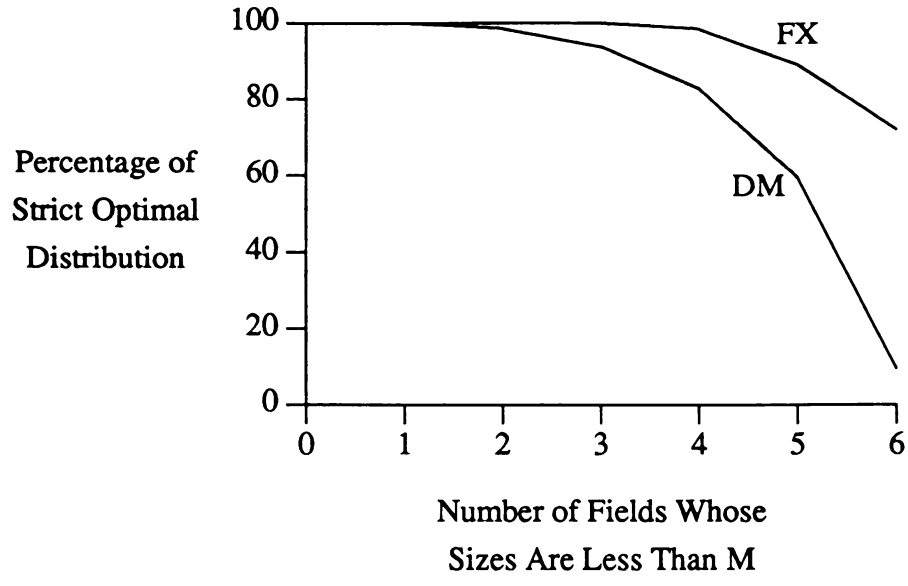


(a) Number of Fields = 6

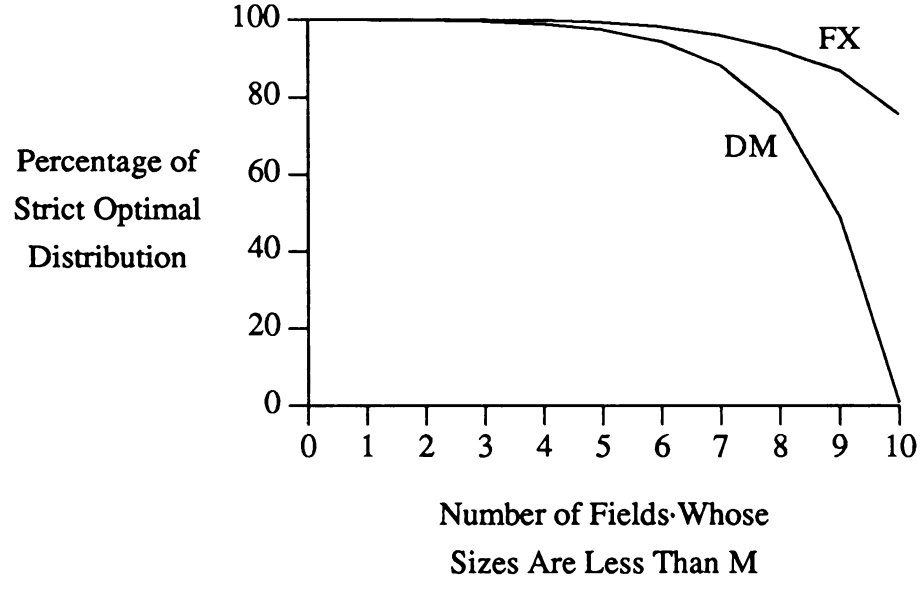


(b) Number of Fields = 10

Figure 3.8. Strict Optimality When for Any Fields r and s, $F_r F_s \geq M$



(a) Number of Fields = 6



(b) Number of Fields = 10

Figure 3.9. Strict Optimality When for Any Fields r, s and t, $F_r F_s F_t \geq M$

Table 3.1 through 3.8 show the largest response size of the DM, GDM and FX distribution methods for various file sizes with various number of parallel devices. The number of fields is six for all these experiments. In all these tables, the first column denotes the number of unspecified fields.

For the GDM method, in order for comparison to be fair, we used seven different sets of multiplication parameters. These sets are GDM1 : 3, 11, 23,37, 49, 53 and GDM2 : 5, 9, 31, 37, 53, 59 and GDM3 : 41, 43, 47, 51, 53, 57 and GDM4 : 3, 5, 7, 11, 13, 17 and GDM5 : 3, 7, 13, 43, 51, 57 and GDM6 : 2, 3, 5, 7, 11, 13 and GDM7 : 2, 5, 11, 43, 51, 57. Here, the sets of parameters in GDM1, GDM2, GDM3, GDM4 and GDM5 are chosen based on [DuS82], i.e., relative prime to the given number of devices. GDM6 and GDM7 are used to include other cases.

For FX distribution methods, field transformation functions applied in each experiment are as follows.

Table 3.1 : <I, U, IU_2, IU_3, I, IU_1>, Table 3.2 : <U, IU_3, IU_4, I, IU_1, IU_2>,

Table 3.3 : <I, U, IU_1, I, U, IU_1>, Table 3.4 : <I, U, IU_1, I, U, IU_1>,

Table 3.5 : <IU_4, U, IU_3, I, IU_1, IU_2>, Table 3.6 : <U, IU_1, IU_3, IU_4, I, IU_2>,

Table 3.7 : <U, IU_3, IU_4, I, IU_1, IU_2> and Table 3.8 : <I, U, IU_2,I, U, IU_2>.

Here, the sequence of transformation functions denotes the sequence of fields to which these transformation functions are applied. Heuristics based on the theorems in section 3.3 and 3.4 are used to choose these transformation functions.

In all these tables, each entry is computed as an average value of largest response sizes from all possible partial match queries for that entry. The tables show that except for first row out of table 3.4 and 3.8, FX distribution methods give smaller largest-response-size than all the other methods. FX distribution is also very close to optimal. It should also be noted that there may exist a set of multiplication parameters by which the GDM method can give better performance than those of GDM1, GDM2, GDM3,

Table 3.1. Response Time for $F_1=F_2=F_3=F_4=2$, $F_5=F_6=4$ and M = 16.

	DM	GDM1	GDM2	GDM3	GDM4	GDM5	GDM6	GDM7	FX	Optimal
2	2.1	1.3	1.7	1.4	1.3	1.5	1.3	1.5	1.1	1.0
3	4.4	2.2	3.1	2.2	2.2	2.4	2.3	2.6	1.6	1.2
4	10.3	4.4	6.2	4.3	4.2	4.7	4.5	5.1	3.0	2.7
5	22.3	8.7	12.3	8.2	8.2	9.2	9.0	10.7	6.7	6.7
6	52.0	18.0	26.0	17.0	18.0	19.0	20.0	22.0	16.0	16.0

Table 3.2. Response Time for $F_1=F_2=F_3=2$, $F_4=F_5=F_6=4$ and M = 32.

	DM	GDM1	GDM2	GDM3	GDM4	GDM5	GDM6	GDM7	FX	Optimal
2	2.4	1.2	1.5	1.3	1.2	1.3	1.2	1.3	1.0	1.0
3	5.7	1.9	2.9	2.1	1.9	2.2	1.9	2.2	1.5	1.1
4	14.8	3.8	6.0	3.8	3.5	4.2	3.5	3.9	2.9	2.2
5	36.0	8.2	13.3	7.8	7.5	8.8	7.8	8.7	6.6	6.0
6	92.0	18.0	28.0	18.0	18.0	20.0	19.0	20.0	16.0	16.0

Table 3.3. Response Time for $F_1=F_2=F_3=F_4=F_5=F_6=8$ and M = 32

	DM	GDM1	GDM2	GDM3	GDM4	GDM5	GDM6	GDM7	FX	Optimal
2	8.0	3.8	4.5	3.7	3.4	3.9	3.3	3.6	3.2	2.0
3	48.0	19.2	21.9	18.9	18.3	20.2	18.1	18.9	16.0	16.0
4	344.0	133.8	143.3	132.5	131.6	138.0	130.5	132.7	128.0	128.0
5	2460.0	1034.7	1058.3	1031.7	1029.3	1045.3	1026.3	1029.7	1024.0	1024.0
6	18152.0	8210.0	8292.0	8202.0	8200.0	8224.0	8196.0	8198.0	8192.0	8192.0

Table 3.4. Response Time for $F_1=F_2=F_3=F_4=F_5=F_6=8$ and M = 64.

	DM	GDM1	GDM2	GDM3	GDM4	GDM5	GDM6	GDM7	FX	Optimal
2	8.0	2.4	2.8	2.4	2.1	2.9	2.1	2.2	2.4	1.0
3	48.0	10.7	11.8	10.6	9.9	12.4	10.2	10.3	8.0	8.0
4	344.0	69.0	73.3	67.5	67.1	75.4	68.3	68.1	64.0	64.0
5	2460.0	522.3	531.3	517.3	516.2	538.2	520.5	517.0	512.0	512.0
6	18152.0	4115.0	4148.0	4102.0	4102.0	4146.0	4114.0	4102.0	4096.0	4096.0

Table 3.5. Response Time for $F_1=2$, $F_2=F_3=4$, $F_4=F_5=F_6=8$ and M = 128.

	DM	GDM1	GDM2	GDM3	GDM4	GDM5	GDM6	GDM7	FX	Optimal
2	4.1	1.2	1.3	1.3	1.2	1.2	1.0	1.4	1.0	1.0
3	17.8	2.8	3.1	2.7	2.5	3.0	2.6	2.9	1.9	1.5
4	81.9	9.6	9.2	8.7	8.2	9.9	8.9	8.8	6.5	6.3
5	351.3	35.3	33.5	32.8	32.2	35.0	35.8	31.8	29.3	29.3
6	1456.0	142.0	134.0	133.0	132.0	135.0	145.0	131.0	128.0	128.0

Table 3.6. Response Time for $F_1=F_2=F_3=F_4=4$, $F_5=F_6=8$ and M = 256.

	DM	GDM1	GDM2	GDM3	GDM4	GDM5	GDM6	GDM7	FX	Optimal
2	4.3	1.1	1.1	1.2	1.0	1.1	1.1	1.1	1.0	1.0
3	17.6	1.8	2.0	1.9	2.2	1.8	2.6	1.9	1.4	1.0
4	79.2	5.2	5.1	4.8	6.7	5.0	8.2	4.9	3.7	2.7
5	352.0	18.2	17.3	16.8	26.7	17.5	34.2	17.3	14.7	13.3
6	1592.0	73.0	72.0	70.0	119.0	71.0	155.0	69.0	64.0	64.0

Table 3.7. Response Time for F_1=F_2=F_3=4, F_4=F_5=F_6=8 and M = 512.

	DM	GDM1	GDM2	GDM3	GDM4	GDM5	GDM6	GDM7	FX	Optimal
2	4.8	1.0	1.0	1.1	1.0	1.0	1.1	1.0	1.0	1.0
3	22.8	1.6	1.8	2.0	2.6	1.6	3.2	1.6	1.4	1.0
4	114.8	4.4	4.8	6.2	9.6	4.5	12.3	4.3	3.5	2.2
5	569.0	15.8	17.3	25.8	44.8	16.0	58.3	15.5	13.3	12.0
6	2848.0	70.0	75.0	122.0	220.0	70.0	289.0	69.0	64.0	64.0

Table 3.8. Response Time for F_1=F_2=F_3=8, F_4=F_5=F_6=16 and M = 512.

	DM	GDM1	GDM2	GDM3	GDM4	GDM5	GDM6	GDM7	FX	Optimal
2	9.6	1.3	1.4	1.4	1.3	1.3	1.7	1.3	2.3	1.0
3	91.2	5.3	5.7	5.6	7.7	5.5	10.0	5.5	5.1	3.2
4	911.2	39.9	40.1	42.2	70.2	40.5	90.3	40.5	37.3	35.2
5	9076.0	395.5	392.7	408.67	700.2	397.7	909.5	397.3	384.0	384.0
6	90404.0	4129.0	4112.0	4313.0	6969.0	4139.0	9176.0	4144.0	4096.0	4096.0

GDM4 and GDM5 in Table 3.1 through Table 3.8. However, even though such set of parameters exists, those can only be found by trial and error method.

In disk based database systems the computation time is not significant compared to disk access time. But in main memory databases CPU computation time is important. Thus, we will discuss CPU computation time for the FX and GDM distribution method.

We use optimized instruction codes for comparing CPU computation time. In the GDM method we use AND operation to implement modulo function. This is possible because the number of devices is assumed to be a power of 2. In FX distribution method, since the multipliers for U and IU_x transformation are always power of 2, we can substitute multiplication by shift operation. Note that we cannot do this in the GDM method because multipliers in the GDM method are usually chosen from prime or odd numbers. Function T_M is done by AND operation.

In MC68000 processor, computation time of FX methods take much less than that of the GDM method. (In MC68000, XOR takes 8 cpu clock cycles, ADD takes 4 clock cycles, AND takes 4 clock cycles, n bit shift takes 6 + 2n clock cycles. But multiplication takes 70 clock cycles). In intel 80286/80386 processor the ratios of clock cycles between different operations are almost similar to those of MC68000.

For main memory databases FX distribution methods are much faster than the GDM distribution method. The computation time of the DM distribution method is less than that of FX distribution methods, but as is shown in the Table 3.1 through Table 3.8, the DM distribution method is not suitable for a large number of parallel devices.

3.6. Data Construction Methods

In this section we discuss data construction methods for the file which is distributed by FX distribution methods. Data distribution methods determine the amount of access concurrency while data construction methods affect storage characteristics and time for each bucket access. We will present two approaches of data construction based on the

usage of multikey hashed directory. Multikey hashing for a given file with n fields produce a subset of T, where $T = f_1 \times f_2 \times \ldots \times f_n$. As discussed in chapter 2, T can be used as either a real global directory or a virtual global directory.

3.6.1. Data Construction Based on Real Global Directory

We will describe methods of using T as a real global directory. Let GD = $[0..F_1-1, \ldots, 0..F_n-1]$ be a multi-dimensional array in which the range of the i-th dimension is $0..F_i$-1. This is the same range of multikey hashing for field i. Here, GD serves as a real global directory. Each element of GD contains the address of a bucket. When the directory is centralized, the problem is trivial. However, for maximum access concurrency the directory also needs to be distributed among the access nodes. Data construction methods for this case will be investigated in the rest of this section.

FX distribution methods partition multi-dimensional array GD into M subsets, where M is the number of access nodes. Since a directory is also distributed among the nodes, we have to have efficient storage rule to locate the elements of this multi-dimensional array. In other words, for each element of GD we have to define the local address in each device. The similar ideas for distributing the elements of an array have been used in MDA memory [Bat77] and the prime memory system [Law82].

In order to determine the local address efficiently, we need a few techniques which are described below.

Definition 3.15. In a given file system, a Minimal Pivot Set (MPS) is any set of fields in which the size of the cartesian product of those fields is no less than the given number of devices M, and any subset of an MPS is not an MPS.

Let there exist an MPS whose cardinality is no greater than 4 in a given file system (if not, no efficient method of using T as a real global directory has been found). Let ρ be the one of MPS's whose cardinality is minimum, and ρ' be the set of all the remaining

fields. In multidimensional array GD, rearrange fields such that those fields in ρ become rightmost dimensions.

When |ρ| = 2, let n-1 and n be two fields in ρ. Here, field n-1 and n denote the fields which correspond to (n-1)-th and n-th dimensions in GD, respectively. Then, apply I-transformation for field n-1 and U-transformation for field n. When |ρ| = 3, let n-2, n-1 and n be three fields in ρ such that $F_{n-2} \geq F_n \geq F_{n-1}$. Then, apply I, U and IU_2 transformation to field n-2, n-1 and n, respectively. Note that the sequence of fields is important in above two cases. When |ρ| = 1, apply I transformation to the field in ρ.

Let the sequence of elements in GD is based on row-major ordering, i.e, index elements of high dimensions change first. Then, the storage rule of this multi-dimensional array is as follows :

For a bucket $< J_1, \ldots, J_n >$ produced by multikey hashing (the sequence of fields in the given bucket is the same as that in GD, i.e., i-th field denotes i-th dimension in GD),

(1) Device No is determined by the FX distribution methods.

(2) The local address for this bucket in the device is $\sum_{i \in \rho'} J_i \gamma_i + \left\lfloor \left(\sum_{i \in \rho} J_i \lambda_i \right) / M \right\rfloor$, where $\gamma_i = \left\lfloor \left(\prod_{j=i+1}^{n} F_j \right) / M \right\rfloor$, and $\lambda_i = \prod_{j=i+1}^{n} F_j$, if $i \neq n$ and $\lambda_i = 1$, if $i = n$.

The correctness of the local address calculation can be shown by the following proposition.

Proposition 3.2. Let GD′ = [0..F_1−1, . . . , 0..F_n−1] be a multi-dimensional array, where the content of element [J_1, ... , J_n] is $T_M \left(\begin{matrix} n \\ [+] \\ j=1 \end{matrix} (X_j(J_j)) \right)$ which is the same as the device number computed by the FX distribution method. Let S = (a_0, . . . , a_{M-1}, a_M, . . , a_{2M}, . .) be a linear sequence of the contents of GD′ by row-major ordering. Let S_i be a subsequence of S such that S_i = (a_{iM}, a_{iM+1}, . . . ,$a_{(i+1)M-1}$). Then, for any S_i, $i = 0, \cdots, \frac{F_1 \times \cdots \times F_n}{M} - 1$, a set of

elements in S_i is Z_M. (It is assumed that there exists an MPS whose cardinality is no greater than 4.)

Proof: When a cardinality of ρ is one (i.e., there exists a field whose size is no less than the given number of devices), or the size of the cartesian product of fields in ρ is M, it simply follows by Lemma 3.2. For other cases, the arrangement of fields in ρ ensures that sequence as shown below.

(case 1) $|\rho| = 2$ (i.e., $I(f_{n-1})$ [+] $U(f_n)$)

The proof immediately follows by Lemma 3.8.

(case 2) $|\rho| = 3$ (i.e., $I(f_{n-2})$ [+] $U(f_{n-1})$ [+]$IU_2(f_n)$)

Since $F_{n-2} \geq F_n \geq F_{n-1}$, $F_n^2 < M$. Let $F_{n-2}F_{n-1}F_n = AM$ and $F_{n-1}F_n = \frac{M}{D}$. Then, $F_{n-2} = AD$. Let $U(f_{n-1})$ [+] $IU_2(f_n) = R$. By Theorem 3.5, R has $F_{n-1}F_n$ distinct elements which are between 0 to M-1. Let $d_{n-1} = M/F_{n-1}$, i.e., $d_{n-1} = F_nD$. Since $d_{n-1} > F_n$, by Lemma 3.8, $R = \bigcup_{i=0}^{F_n-1} \left[U(f_{n-1}) + c_i\right]$ for F_n different number of c_i's which are between 0 to d_{n-1}. Now it is sufficient to show that there exists exactly one c_i for each interval of size D between 0 to d_{n-1}. This guarantees that for any set W of D consecutive elements in f_{n-2} which are all in the same interval of size D, R[+]$W = Z_M$ by Lemma 3.2.

Now, let $d_{n1} = M/F_n$ and $d_{n2} = M/F_n^2$. We want to show that for any $K_1, K_2 \in f_n$ such that $K_2 > K_1$, $c_i = (K_1[+]K_1d_{n1}[+]K_1d_{n2})$ *mod* d_{n-1} and $c_j = (K_2[+]K_2d_{n1}[+]K_2d_{n2})$ *mod* d_{n-1} are in different intervals of size D. Let $d_{n-1} = Bd_{n1}$. When K_1 *mod* $B \neq K_2$ *mod* B, it is easy to see that c_i and c_j cannot be in the same interval of size D because $d_{n1} > D$. Let K_1 *mod* $B = K_2$ *mod* B. This implies that K_1 and K_2 are in the different intervals of size B. Therefore, by Lemma 3.12, $K_1[+]K_1d_{n2}$ and $K_2[+]K_2d_{n2}$ should be in different intervals of size Bd_{n2} between 0 to d_{n-1}. Since Bd_{n2} = D, the proof follows. □

When a multikey hash function $\Theta 1$ is used for a file V, using T as a real global directory is advantageous if for most $t \in T$, $\Theta 1^{-1}(t) \in V$ with reasonable average chain length. However, an appropriate $\Theta 1$ may not be easily determined for dynamic files because the file size is not known in advance. Hence, the real global directory may turn out to be very sparse or to have long overflow chain. Here, it should be noted that applying dynamic hash function for $\Theta 1$ will cause significant overhead due to internode data movement. The virtual global directory approach described in the next section can avoid this problem.

3.6.2. Data Construction Based on Virtual Global Directory

In the previous section we described the method of using T as a real global directory. In this section we describe the method of using T as a virtual global directory. The idea of the virtual global directory is to use one more hash function $\Theta 2$ which is local in each device. Let $< J_1, \ldots, J_n >$ be an ordered n-tuple produced by multikey hash function $\Theta 1$ for some record. The local hash function $\Theta 2$ uses this ordered n-tuple as an input key for its local directory. Since $\Theta 2$ does not affect data distribution, dynamic hash functions [Fag79, Lar78, Lit80] can be used as $\Theta 2$. When T is used as a virtual global directory, only the local directories physically exist. Each local directory can dynamically grow and shrink while the virtual global directory is static.

This two-level mapping data construction is much more flexible than the real global directory because the storage utilization of directories is not affected by $\Theta 1$, and dynamic hash function for local directories can handle dynamic files. Thus, a virtual global directory approach is appropriate for dynamic files.

One disadvantage of the virtual global directory approach is that it may cause more probings to find qualified records than in the real global directory. This is because different ordered n-tuples produced by $\Theta 1$ can be mapped into the same local directory entry by $\Theta 2$.

Let V1 be the set of ordered n-tuples produced by $\Theta 1$ for a given file which are allocated into the same device, and V2 be the range of local hash function $\Theta 2$ in that device. Let $\mu 1 = |V1|$ and $\mu 2 = |V2|$. Let τ be an average number of elements in V1 which are mapped into the same $v \in V2$ by $\Theta 2$. Then, the probability P that $\Theta 2^{-1}(v) = \phi$ for $v \in V2$, is given by $P = \left[1 - \frac{1}{\mu 2}\right]^{\mu 1}$. Let $c = \mu 2/\mu 1$. Then, $\tau \approx \frac{1}{c(1-e^{-1/c})}$. This can be derived by using $\lim_{\mu 1 \to \infty} (1 - \frac{1}{c\mu 1})^{\mu 1} = e^{-1/c}$, and $\tau = \frac{\mu 1}{(1-P)\mu 2}$. For example, $\tau = 1.58$ when $c = 1$, and so the number of probings increases. (Detailed description about average chain length for various hashing algorithms can be found in [Knu73].)

However, since T consists of cartesian product of all fields, many elements in T may not correspond to any record. On the other hand, the two level hashing scheme (i.e., the virtual global directory approach) can always achieve efficient storage utilization of the directories because the input of $\Theta 2$ is only those n-tuples which have corresponding records. Even for the case when a real global directory has many empty entries, the two level hashing scheme can always guarantee efficient storage utilization with slightly increased number of probings.

CHAPTER 4

OPTIMAL DATA DISTRIBUTION FOR MULTIATTRIBUTE RANGE QUERIES

4.1. Introduction

In this chapter we describe optimal data distribution for multiattribute range queries. A *multiattribute range query*, also called a *region query* or *orthogonal range query*, is an intersection query in which a multiplicity of the attributes are allowed to be range specified in a query's qualification. The multiattribute range query differs from the partial match query in that the range specification is allowed in the multiattribute range query while it is not allowed in the partial match query.

Several file structures have been proposed for handling multiattribute range queries (hereafter, a range query denotes a multiattribute range query). These include various types of specialized tree structures and hash based accesses. In [Ben75] the multidimensional binary search tree, also called *k-d tree*, has been proposed, which is an extension of the standard binary search tree. Worst case analyses of these trees can be found in [Lee77]. There are other types of the tree structure for range searching [Ben79]. Hash based accesses for range queries has also been investigated in the past. In [Bol81], Bolour proposed box-array addressing functions which is a composition of randomizing function and order-preserving function. It has been shown in that paper that this hashing scheme is effective for answering small range queries. Several other types of the file structure for range searching can also be found in [Ben79, Knu73]. A lower bound on the complexity of range queries has been presented in [Fre81]. The main focus of those research is on minimizing the number of bucket accesses. However, they did not consider data distribution to enhance access concurrency. Though there are

some heuristic data distribution methods for multikey search queries [DuS82, Kim88], range specification in a query's qualification is not considered in those methods. In this chapter we will mainly focus on optimal data distribution for multiattribute range queries to facilitate parallel processing of this type of applications.

A partial match query can be thought of as a special case of a range query because a range query reduces to a partial match query when each specified range size is only one. Thus, a partial match query will be treated as a special case of a range query in the rest of this chapter.

The rest of this chapter is organized as follows. In section 4.2 we describe underlying file structure, basic definitions and assumptions. In section 4.3 we show the inherent limitations for optimal data distribution for some types of range queries. We describe optimal data distribution methods for range queries in section 4.4. The optimality conditions for these methods are also given.

4.2. Definitions And Terminology

In this section we describe underlying file structures as well as relevant definitions and assumptions.

We use partitioned hashed directory as a file structure. In other words, the access structure for a file with n secondary keys is a partitioned hashed directory consisting of n fields. This directory is based on multikey hashing scheme, where each hash function is order-preserving. Multikey hash function is described in chapter 3. Several order-preserving hash functions have been proposed in [Gar86,Hsi88]. In the following paragraph we also give a method for implementing order-preserving hashed accesses by small tables.

One way of implementing order-preserving hash functions is to represent the function explicitly by a table. Figure 4.1 shows the explicit representation of n functions for a file consisting of n fields, where h_i denotes the function (i.e., table to show the function

values) applied to i-th attribute values. Each table of the figure the first column denotes the interval of key values and the second column denotes the image for any domain value in that interval.

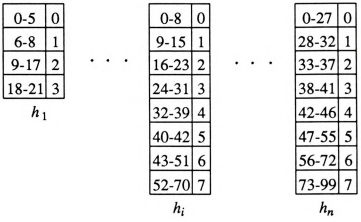


Figure 4.1. Explicit Representation of Multikey Hash Functions

In this structure a file consisting of n fields (i.e., n secondary keys) needs n tables. Let f_i be the projection of the second column of table i, i.e., $f_i = \{0, 1, \ldots, 7\}$ in Figure 4.1. A bucket is defined as an ordered n-tuple, $<b_1, \ldots, b_n>$, such that b_i is an element of f_i. Then, the cartesian product $f_1 \times \ldots \times f_n$ represents the multikey hash directory. As we discussed in chapter 3, this directory can be used as either a global directory or a virtual global directory.

This representation requires additional storage for the tables as in Figure 4.1. Let us consider the storage overhead of these tables for the file with five fields. Suppose the directory consists of thirty thousand (about 2^{15}) entries (if a bucket contains 33 records in average, the file contains about one million records). When all the field sizes are the same, the size of each field is 8. If each entry of the tables takes ten bytes, the storage overhead is only 400 bytes because we need 5 * 8 = 40 entries altogether. Thus, the storage overhead is negligible and these tables can be easily stored in main memory.

We use a bucket as a unit of data distribution. The main focus of this chapter is to investigate optimal bucket distribution for range queries in multikey hashing. Since we are dealing with multikey hash file which is the same as that in chapter 3, all the terminology defined in chapter 3 are also used in this chapter. In the rest of this section we describe definitions which are necessary in this chapter.

Definition 4.1. r(u, v) denotes the range specification between u and v, where both boundaries are closed.

We assume that a range query can have three types of field value specification which are single value, range, and don't care in which case it is denoted by *. We will not use r(u, v) to denote a single value (i.e., range size is equal to one) and don't care (i.e., unspecified). We also do not allow cyclic ranges like r(5,2), i.e., 5, 6 , . . . , F_i−1, 0, 1, 2. In other words, whenever r(u, v) is a range specification for field i, it is implied that $u < v$ and, either $u \neq 0$ or $v \neq F_i - 1$.

Definition 4.2. A range query for files with n fields is denoted by $[A_1, A_2, \ldots, A_n]$, where for each $i = 1, \ldots, n$, A_i = *, or r(u, v), or w, where $u, v, w \in f_i$ and $u < v$.

We define types of range queries based on the number of range specified fields in a query's qualification.

Definition 4.3. When the number of range specified fields is α in a query q, query q is called *type* α range query.

Example 4.1. Let q_1, q_2 and q_3 be queries in the file (DEPT, AGE, STATE) such that q_1 = [Math, r(20, 27), Ohio], q_2 = [*, r(20, 27), *] and q_3 = [r(Math, Physics), r(20, 27), *]. Here, r(20, 27), r(Math, Physics) denote range specification, and "*" denotes don't care condition, i.e., the field value is unspecified. Then, q_1 and q_2 are type 1 range queries, and q_3 is type 2 range query. By the above definition a partial match query such as [Math, *, Ohio] is a type 0 range query.

Definition 4.4. Let $R(q)$ be the set of buckets which satisfy the qualification for a range query q. The distribution method is called *strict optimal* for a range query q in a given file system if each device has no more than $\lceil |R(q)|/M \rceil$ number of buckets.

Definition 4.5. When the distribution method is strict optimal for all queries in type 0 through type α range queries in a given file system, it is called *perfect optimal* for type $(0 - \alpha)$ range queries in that file system.

Example 4.2. Let $f_1 = \{0, 1, 2, 3\}$, $f_2 = \{0, 1, 2, 3\}$ and M = 8. Figure 4.2 shows three bucket distributions denoted by Distribution-1, Distribution-2 and Distribution-3. For example, the bucket <3, 1> is stored in device 5 for Distribution-1, device 2 in Distribution-2 and device 7 in Distribution-3.

f_1	f_2	Distribution-1	Distribution-2	Distribution-3
0	0	0	0	0
0	1	1	1	4
0	2	2	2	2
0	3	3	3	6
1	0	4	4	1
1	1	5	5	5
1	2	6	6	3
1	3	7	7	7
2	0	0	7	2
2	1	1	6	6
2	2	2	5	0
2	3	3	4	4
3	0	4	3	3
3	1	5	2	7
3	2	6	1	1
3	3	7	0	5

Figure 4.2. Example Bucket Distributions

Let $q_1 = [r(1, 2), *]$ and $q_2 = [*, r(0, 1)]$. Then, the Distribution-1 is strict optimal for query q_1, but is not strict optimal for query q_2. The Distribution-2 is strict optimal for query q_2, but is not strict optimal for query q_1. The Distribution-3 is perfect

optimal for type (0 - 1) range queries, and hence it is strict optimal for both queries q_1 and q_2. However, this distribution is still not perfect optimal for type (0 - 2) range queries because it is not strict optimal for query q_3 = [r(0, 2), r(0, 2)].

Definition 4.6. Let a file consists of n fields, and S_i, $i = 1, \ldots, n$, be a subset of f_i. Then, $M(S_1, \ldots, S_n)$ denotes a set of devices at which bucket $<b_1, \ldots, b_n>$is stored, where $b_i \in S_i$. For convenience, when $S_i = \{s_i\}$, i.e., S_i has a single element, we will use s_i instead of $\{s_i\}$.

For example, in Distribution-1 of Figure 4.2, M(0, 0) = {0}, M({0, 1}, 0) = {0, 4} and M({0, 1, 2}, 0) = {0, 4}.

Since the perfect optimal distribution is the most desirable, it is worth investigating whether a perfect optimal distribution always exists. It will be shown that perfect optimal distribution for certain types of range queries does not exist inherently in many cases. In the following section we discuss nonexistence of perfect optimal distribution for certain types of range queries. We give data distribution methods for optimal bucket distribution for various types of range queries in later sections. It will be shown that the proposed data distribution methods are perfect optimal for certain types of range queries, and strict optimal for a large class of range queries.

4.3. Limitations of Perfect Optimal Distribution

In this section we will show that there are inherent limitations for achieving perfect optimal distribution for certain types of range queries. In section 4.3.1 we show that for files with two or more fields, perfect optimal distribution for type (0 - 2) range queries does not exist in many cases. In section 4.3.2 it will be shown that for files with three or more fields, perfect optimal distribution for type (0 - 1) range queries is not always possible. In both sections the sufficient conditions for the nonexistence of perfect optimal distribution will be given. In section 4.3.3 we discuss perfect optimal

distribution for type 0 range queries.

4.3.1. Type (0 - 2) Range Queries

In this section we show that perfect optimal distribution for type (0 - 2) range queries does not exist in many cases.

Lemma 4.1. When a file consists of n fields ($n \geq 2$) and the given number of devices is M, perfect optimal distribution for type (0 - 2) range queries does not exist if there are at least two fields i and j, and two integers a and b such that $2 \leq a < F_i$, $3 \leq b \leq F_j$ and ab = M.

Proof: It is sufficient to prove the lemma for files consisting of only two fields. Assume perfect optimal distribution for type (0 - 2) range queries exists. Thus, we have a distribution for this file system which is perfect optimal for type (0 - 2) range queries. Let S = $\{s_1, \ldots, s_a\}$ be a subset of f_i, where $a \geq 2$ and $s_1 > 0$, $s_2 = s_1+1, \ldots, s_a = s_{a-1}+1$, and T = $\{t_1, \ldots, t_b\}$ be a subset of f_j, where $b \geq 3$ and $t_2 = t_1+1, \ldots, t_b = t_{b-1}+1$ such that ab = M. In other words, S and T are a and b consecutive values in field i and j, respectively, where |S × T| = M. Clearly, when A and B are sets of consecutive elements in field i and j, respectively, such that |A × B| < M, M(A, B) is a set whose size is |A × B|. Thus, the sets M(s_1, T), . . . , $M(s_a, T)$ are mutually disjoint and $M(S, T) = Z_M$ (otherwise, the distribution is not strict optimal for query q_1 = [r(s_1, s_a), r(t_1, t_b)]). Since the sets $M(s_1-1, T), \ldots, M(s_a-1, T)$ are also mutually disjoint, and the size of cartesian product of these sets is M, $M(s_1-1, T) = M(s_a, T)$ (Figure 4.3 gives a pictorial view of these steps). We will show a contradiction for either case when the size of T is even, or odd.

(case 1) b is even, (i.e., $b \geq 4$)

Let $T_{11} = \{t_1, \ldots, t_{b/2}\}$, and $T_{12} = \{t_{b/2+1}, \ldots, t_b\}$. The sets $M(s_1-1, T_{11})$ and $M(s_a, T_{11})$ are disjoint (otherwise, the distribution is not strict optimal for query q_2 = [r(s_1−1, s_a), r(t_1, $t_{b/2}$)]. Note that the number of qualified buckets for q_2 is less than

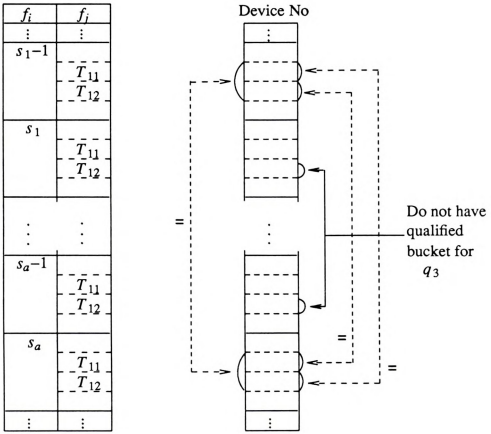


Figure 4.3. Nonexistence of Perfect Optimal Distribution for Type (0 - 2) Range Queries

M). Thus, $M(s_1-1, T_{11})$ is equal to $M(s_a, T_{12})$, and $M(s_1-1, T_{12})$ is equal to $M(s_a, T_{11})$.

Now, let us consider range query $q_3 = [r(s_1-1, s_a), r(t_1, t_{b/2+1})]$. Suppose the bucket $<s_1-1, t_{b/2+1}>$ is stored at device m_0, and the bucket $<s_a, t_{b/2+1}>$ is stored at device m_1. Since $t_{b/2+1} \in T_{12}$ and $M(s_1-1, T_{12}) = M(s_a, T_{11})$, m_0 is an element of $M(s_a, T_{11})$. Similarly, m_1 is an element of $M(s_1-1, T_{11})$. Thus, device m_0 and m_1 have at least two qualified buckets for query q_3. However, the devices in $\bigcup_{k=1}^{a-1} M(s_k, \{t_{b/2+2}, \ldots, t_b\})$ do not have any qualified buckets for query q_3. This contradicts the assumption.

(case 2) b is odd

Let $T_{11}' = \{t_1, \ldots, t_{b'}\}$ and $T_{12}' = \{t_{b'+1}, \ldots, t_b\}$, where $b' = \frac{b+1}{2}$. The sets $M(s_1-1,T_{11}')$ and $M(s_a,T_{11}')$ are disjoint (otherwise, the distribution is not strict optimal for query $q_4 = [r(s_1-1, s_a), r(t_1, t_{b'})]$. Note that the number of qualified buckets for q_4 is less than or equal to M because $(a+1)(\frac{b+1}{2}) \leq ab$ when $a \geq 2$ and $b \geq 3$). However, $M(s_1-1, T) = M(s_a,T)$ and $M(s_1-1,T_{11}') \cap M(s_a,T_{11}') = \phi$ cannot be satisfied at the same time because $|T_{11}'| > |T|/2$. This is a contradiction. □

Example 4.3. This example explains the proof of Lemma 4.1 through an example file system. Let $f_1 = \{0, 1, 2, 3\}$, $f_2 = \{0, 1, 2, 3\}$ and M = 8. Note that this file system satisfies the conditions of Lemma 4.1. Let buckets be distributed as in Figure 4.4, where m_k, $k = 0, \ldots, 15$, denotes an element in Z_8. Since we have only eight devices, $m_u \neq m_v$ is not implied by $u \neq v$ in the table.

Suppose Figure 4.4 is a perfect optimal distribution for type (0 - 2) range queries. Then, $\{m_0, m_1, \ldots, m_7\} = \{m_4, m_5, \ldots, m_{11}\} = Z_8$. Otherwise, the distribution is not strict optimal for at least one of the range queries $q_1 = [r(0, 1), *]$ and $q_2 = [r(1, 2),$

f_1	f_2	Device No
0	0	m_0
0	1	m_1
0	2	m_2
0	3	m_3
1	0	m_4
1	1	m_5
1	2	m_6
1	3	m_7
2	0	m_8
2	1	m_9
2	2	m_{10}
2	3	m_{11}
3	0	m_{12}
3	1	m_{13}
3	2	m_{14}
3	3	m_{15}

Figure 4.4. Bucket Distribution When $F_1 = 4, F_2 = 4$ and M = 8

*]. This implies that $\{m_0, m_1, m_2, m_3\} = \{m_8, m_9, m_{10}, m_{11}\}$. Since $\{m_0, m_1\}$ and $\{m_8, m_9\}$ are disjoint (otherwise, the distribution is not strict optimal for range query q_3 = [r(0, 2), r(0, 1)]), $\{m_0, m_1\} = \{m_{10}, m_{11}\}$ and $\{m_2, m_3\} = \{m_8, m_9\}$. Let us consider the range query q_4 = [r(0, 2), r(0, 2)]. Then, the devices which have the qualified buckets for this range query are $m_0, m_1, m_2, m_4, m_5, m_6, m_8, m_9, m_{10}$. Here, m_2 is one of m_8 or m_9, and m_{10} is one of m_0 or m_1. Thus, at least two devices have more than one qualified buckets for this range query. However, m_7 does not have any qualified bucket for q_3. This contradicts the assumption.

Lemma 4.2. When a file consists of n fields ($n \geq 2$) and the given number of devices M is equal to four, perfect optimal distribution for type (0 - 2) range queries does not exist if there are at least two fields i and j such that $F_i \geq 3$ and $F_j \geq 3$.

Proof: It is sufficient to prove the lemma for files consisting of only two fields. Suppose perfect optimal distribution for type (0 - 2) range queries exists. Thus, we have a

distribution for this file system which is perfect optimal for type (0 - 2) range queries. Let $\{m_0, m_1, m_2, m_3\}$ be the set of four devices. Since any two of buckets <0, 0>, <0, 1> and <0, 2> should not be stored at the same device, let $M(0,0) = \{m_0\}$, M(0, 1) = $\{m_1\}$ and $M(0,2) = \{m_2\}$. Since M(0, {0, 1}) and M(1, {0, 1}) are disjoint (otherwise, the distribution is not strict optimal for query q_1 = [r(0, 1), r(0, 1)]), and M(1, {0, 1}) and M(2, {0, 1}) are disjoint, M(0, {0, 1}) = M(2, {0,1}). Since M(0, 0) is not equal to M(2, 0), M(0, 0) = M(2, 1) and M(0, 1) = M(2, 0). Thus, bucket <2, 0> is stored at device m_1, and bucket <2, 1> is stored at device m_0. Since bucket <2, 2> cannot be stored at any one of the devices m_0, m_1 and m_2 (otherwise, the distribution is not strict optimal for at least one of the range queries q_2 = [2, r(0, 2)] and q_3 = [r(0, 2), 2]), M(2, 2) = $\{m_3\}$.

Now, for any allocation of bucket <1, 1> to any one of the devices m_0, m_1, m_2 and m_3, the distribution is not strict optimal for at least one of the queries q_4 = [r(0, 1), r(1, 2)] and q_5 = [r(1, 2), r(1, 2)]. This is a contradiction. □

Theorem 4.1. When a file consists of n fields ($n \geq 2$) and the given number of devices is M, there does not exist a perfect optimal distribution for type (0 - 2) range queries if (1) there are at least two fields i and j, and two integers a and b such that $2 \leq a < F_i$, $3 \leq b \leq F_j$ and $ab = M$, or (2) there are at least two fields i and j such that $F_i \geq 3$, $F_j \geq 3$ and $M = 4$.

Proof: This is a direct consequence of Lemma 4.1 and Lemma 4.2. □

Thus, there does not exist a data distribution method which guarantees perfect optimal distribution for type (0 - 2) range queries even for every file with only two fields.

4.3.2. Type (0 - 1) Range Queries

In this section we show that for files with three or more fields, there does not exist perfect optimal distribution for type (0 - 1) range queries in many cases.

Theorem 4.2. When a file consists of n ($n \geq 3$) fields and the given number of devices is M, there does not exist a data distribution which is strict optimal for any type 0 range query that has at most two unspecified fields, and for any type 1 range query that has at most one unspecified field if (1) there are at least three fields i, j and k such that $F_i \leq F_j < F_k$, and $M < F_iF_k \leq F_jF_k \leq 2M$, and $F_k \geq F_j + F_jF_k / M$, and (2) there are two integers b and c such that $bF_j = cF_k = M$.

Proof: It is sufficient to prove the theorem for files consisting of only three fields. For convenience, let these three fields be 1, 2 and 3 such that $F_1 \leq F_2 < F_3$ and $M < F_1F_3 \leq F_2F_3 \leq 2M$ and $F_3 \geq F_2 + F_2F_3 / M$. Suppose the theorem is not true. Then, we have a distribution which is strict optimal for any query in the theorem. Let $cF_3 = M$ and $bF_2 = M$. This implies that $c < F_1$ and $b < F_3$. Suppose bucket <0, 0, 0> is stored at device m_0, bucket <0, 0, 1> is stored at $m_1, \ldots,$ and bucket <0, 0, F_3-1> is stored at m_{F_3-1}. Clearly, for any $u \in f_1$, $v \in f_2$ and a set W which contains some consecutive elements in f_3 such that $|W| < M$, $M(u,v,W)$ is a set whose size is $|W|$, and hence $m_0, \ldots, m_{F_3-1}$ are all different. We will show that buckets <0, c, b> and <c, 0, b> should be stored at device m_0. At first, M(0, 0, f_3) is equal to M(0, c, f_3) (otherwise, the distribution is not strict optimal for at least one of the range queries q_1 = [0, r(0, c-1), *] and q_2 = [0, r(1, c), *]). Thus, the possible set of devices at which bucket <0, c, b> can be stored is $\{m_0, \ldots, m_{F_3-1}\}$. Bucket <0, c, b> cannot be stored at any one of the devices $m_1, \ldots, m_b$. Otherwise, for range query q_3 = [0, *, r(1, b)], at least one of the devices $m_1, \ldots, m_b$ has two or more qualified buckets while there exist a device which does not have any qualified bucket. Note that the number of qualified buckets for q_3 is M. Bucket <0, c, b> cannot also be stored at any one of the devices $m_{b+1}, \ldots, m_{F_3-1}$. Otherwise, the distribution is not strict optimal for range query q_4 = [0, *, r(b, F_3-1)]. Note that the number of qualified buckets for q_4 is less than or equal to M. Thus, bucket < 0, c, b> should be stored at device m_0. (Figure 4.5 gives a pictorial view of these steps.)

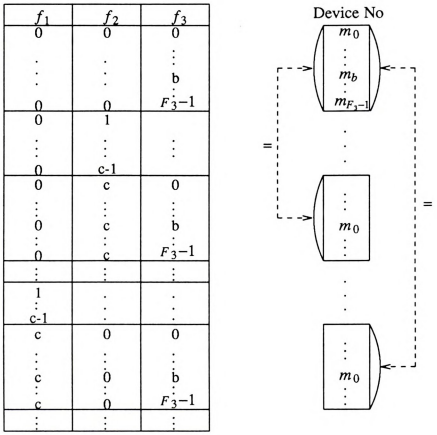


Figure 4.5. Nonexistence of Perfect Optimal Distribution for Type (0 - 1) Range Queries

Now, in order for the distribution to be strict optimal for queries q_5 = [r(0, c-1), 0, *] and q_6 = [r(1, c), 0, *], M(c, 0, f_3) should be the same as M(0, 0, f_3). Thus, the possible set of devices at which bucket <c, 0, b> can be stored is $\{m_0, \ldots, m_{F_3-1}\}$. Bucket <c, 0, b> cannot be stored at any one of the devices $m_1, \ldots, m_{F_3-1}$. Otherwise, the distribution is not strict optimal for at least one of the queries q_7 = [*, 0, r(1,b)] and q_8 = [*, 0, r(b,F_3−1)]. Thus, bucket <c, 0, b> should be stored at device m_0. However, this bucket distribution is not strict optimal for query q_9 = [*, r(0, c), b] because device m_0 has at least two qualified buckets for q_9 while there exists a device which does not have any qualified bucket for q_9. Note that cF_3 = M and $F_3 > F_2 + F_2F_3$ / M imply that the number of qualified buckets for q_9 is less than or equal to M. This completes the proof. □

Corollary 4.1. When a file consists of n ($n \geq 3$) fields and the given number of devices is M, perfect optimal distribution for type (0 - 1) range queries does not exist if the same conditions of Theorem 4.2 are satisfied.

Corollary 4.2. When a file consists of n ($n \geq 3$) fields and the given number of devices is M, there does not exist a data distribution which is strict optimal for any range query that has at most one range specified and at most one unspecified field if (1) there are at least three fields i, j and k such that $F_i < F_j < F_k$, and $M < F_iF_k \leq F_jF_k \leq 2M$, and $F_k \geq F_j + F_jF_k$ / M, and (2) there are two integers b and c such that $bF_j = cF_k = M$.

Proof: It can be observed that Theorem 4.2 is proved by considering range queries (q_1 through q_9) which have at most one unspecified field. The only one possible exception is query q_9 which may have two unspecified fields if $F_j = c + 1$. When the condition $F_i \leq F_j < F_k$ is replaced by $F_i < F_j < F_k$ from those of Theorem 4.2, the query q_9 is guaranteed to be a type 1 range query (i.e., r(0, c) is a range specification) because $F_j > F_i \geq c + 1$. Thus, the corollary follows. □

By Corollary 4.2, there does not exist a data distribution method which guarantees strict

optimal distribution for any range query that has at most one range specified and at most one unspecified field even for every file with only three fields. This also implies that there is no data distribution method which guarantees perfect optimal distribution for type (0 - 1) range queries for every file with only three fields.

4.3.3. Type 0 Range Queries

When each field size and the given number of devices are power of 2, by Corollary 3.4 perfect optimal distribution for type 0 range queries is always possible if the number of fields whose sizes are less than the given number of devices, is no greater than four. The data distribution methods for type 0 range queries are also presented in chapter 3. In [Sun87] it has been shown that perfect optimal distribution does not exist for binary cartesian product files with n fields, if $M \geq 4$ and $n \geq \lfloor \log_2 M \rfloor + 2$. This result implies that perfect optimal distribution for type 0 range queries is not always possible for files with four or more fields. However, for general file systems, sufficient condition for either existence or nonexistence of perfect optimal distribution for type 0 range queries has not been found.

We have shown through Theorem 4.1 and Theorem 4.2 that there are inherent limitations to achieve perfect optimal distribution for range queries even for a file consisting of small number of fields. In the following sections we present optimal data distribution methods for range queries. These methods are the extended version of FX distribution methods presented in chapter 3.

4.4. Optimal Data Distribution Methods for Range Queries

In this section we present FX distribution methods for range queries. We describe basic optimality conditions in section 4.4.1. In section 4.4.2 and the following sections we will present field transformation functions developed for range queries. The conditions to achieve optimal distribution by these field transformation techniques will also be

described. For convenience we assume, from now on, that each field size and the given number of devices M are power of 2.

4.4.1. Optimal Distribution by Basic FX Distribution Method

In this section we describe conditions for optimal distribution for various types of range queries by using the Basic FX distribution method. The definition of the Basic FX distribution method is given in section 3.3.

Example 4.4. Figure 4.6 shows the bucket distribution by the Basic FX distribution method, where f_1 = {0, 1, 2, 3}, f_2 = {0, 1, 2, 3} and M = 4. Here, Device No = $T_M \left[J_1 [+] J_2\right]$, where $J_1 \in f_1$, $J_2 \in f_2$ and T_M returns the rightmost two bits of the result of $J_1 [+] J_2$.

f_1	f_2	Device No
000	000	0
000	001	1
000	010	2
000	011	3
001	000	1
001	001	0
001	010	3
001	011	2
010	000	2
010	001	3
010	010	0
010	011	1
011	000	3
011	001	2
011	010	1
011	011	0

Figure 4.6. Basic FX Distribution When $F_1 = F_2 = 4$ and M = 4.

We can verify that the distribution of Figure 4.6 is perfect optimal for type (0 - 1) range queries but is not perfect optimal for type (0 - 2) range queries because it is not

strict optimal for query [r(0, 1), r(0, 1)]. In fact, by Theorem 4.1 there does not exist a perfect optimal distribution for type (0 - 2) range queries for this file system.

Since we have shown that FX distribution methods are strict optimal for mosr class of type 0 range queries in chapter 3, we will not discuss the cases when only type 0 range queries are involved.

Lemma 4.3. For any type 0 range query which has at least one unspecified field whose size is greater than or equal to M, all the devices have equal number of qualified buckets by the Basic FX distribution method.

Proof: The proof is almost the same as that of Theorem 3.2. □

Theorem 4.3. The Basic FX distribution method is strict optimal for any range query, for which there exists at least one unspecified field whose size is greater than or equal to the given number of devices M.

Proof: Since a range query is a set of partial match (i.e., type 0 range) queries, and by Lemma 4.3 we know that equal number of qualified buckets are distributed in all the devices for each partial match query, the proof immediately follows. □

Corollary 4.3. The Basic FX distribution method is strict optimal for any range query, for which there exists at least one range specified field such that the specified range size is an integral multiple of the given number of devices M.

Proof: When the specified range size is F_i, it is easy to see that the effect of the Basic FX distribution method for this range is the same as that of the Basic FX distribution method for an unspecified field whose size is F_I (note that for any integers J_1 and J_2, $T_M(J_1 \,[+]\, J_2) = T_M(J_1)\,[+]\,T_M(J_2)$). Thus, the corollary follows. □

Theorem 4.4. The Basic FX distribution method is strict optimal for any range query, for which there is at most one range specified field and all the other fields are specified as single values.

Proof: The proof is almost the same as that of Theorem 3.1. □

Theorem 4.5. When all the field sizes are greater than or equal to the given number of devices M, the Basic FX distribution method is perfect optimal for type (0 - 1) range queries.

Proof: When all the fields are specified as a single value, the proof is trivial. When at least one field is unspecified, the proof follows by Theorem 4.3. The only remaining case is when one field is specified as a range and all the other fields are specified as single values. The proof for this case follows by Theorem 4.4. □

Theorem 4.3, 4.4 and 4.5 show sufficient conditions for optimal distribution by the Basic FX distribution method. However, the Basic FX distribution method does not give optimal distribution for many types of range queries. The following proposition gives the conditions for optimal distribution for these cases.

Proposition 4.1. Let $q(f) = \{i_1, i_2, \ldots, i_k\}$ be the set of range specified or unspecified fields for a range query q. Let S_{i_j} be the set of elements in the range when field i_j is range specified, or be f_{i_j} when field i_j is unspecified. Then, FX distribution methods are strict optimal for a range query q, if there exists a set of fields $\{i_1, \ldots, i_j\} \subseteq q(f)$ such that $|S_{i_1} \times \cdots \times S_{i_j}|$ is an integral multiple of M, and $\#\{(J_{i_1}, \cdots J_{i_j}) \in S_{i_1} \times \cdots \times S_{i_j} \mid T_M \left[\sum_{p=1}^{j} (J_{i_p})\right] = z\} = |S_{i_1} \times \cdots \times S_{i_j}|/M$ for all $z \in Z_M$.

Proof: The proof is similar to that of Theorem 3.2. □

Let S_i be the set of elements in the range when field i is range specified, or be f_i when field i is unspecified. Proposition 4.1 says that we can guarantee strict optimal distribution for a given range query, if

(1) there exists a subset of the range specified or unspecified fields $\{i_1, \ldots, i_j\}$ such that $|S_{i_1} \times \cdots \times S_{i_j}|$ is an integral multiple of M, and

(2) the records projected on these sets of fields are distributed uniformly among the M devices.

In other words, optimal distribution for a subset of fields guarantees strict optimal distribution for many queries in which those fields are unspecified or range specified.

However, when the size of none of the fields is greater than or equal to M, the conditions given in Proposition 4.1 are not satisfied in the Basic FX distribution method. Thus, in the next section we propose field transformation techniques for the fields whose sizes are less than the given number of devices M. Note that the definition of field transformation functions is given in section 3.4. These field transformation techniques increase the scope of optimality over that of the Basic FX distribution method. By the definition of field transformation functions (i.e., one-to-one function), it is easy to see that all the lemmas, theorems and proposition that hold for the Basic FX distribution method also hold for FX distribution methods (i.e., the Basic FX distribution method with field transformation functions).

4.4.2. Field Transformation Functions for Range Queries

In the previous section conditions for optimal distribution have been described when the Basic FX disribution method is used. In this section we present field transformation techniques which can improve the performance over the Basic FX distribution method significantly. The following paragraph exemplifies the idea.

When $f_1 = \{0, 1, 2, 3\}$, $f_2 = \{0, 1, 2, 3\}$ and M = 8, the disribution by the Basic FX distribution method is not strict optimal for many range queies (we can easily see that the distribution in Figure 4.6 is not strict optimal for many range queries when M = 8). Suppose X is an one-to-one mapping such that $X(0) = 0$, $X(1) = 4$, $X(2) = 2$, $X(3) = 6$. When each bucket $<u, v>$ in the file $f_1 \times f_2$ is stored at the device u [+] $X(v)$, the distribution is strict optimal for any range query in which field 1 is not specified as a range. (It can be easily verified by substituting $(100)_B$, $(010)_B$ and $(110)_B$ for $(001)_B$, $(010)_B$ and $(011)_B$, respectively in f_2 column of Figure 4.6. In fact this distribution is perfect optimal for type (0 - 1) range queries.) Thus, our objective is to find such mapping, X

in general, for the given file system. It will be shown that for any values of F_1, F_2 and M, such mapping can be easily found. The following proposition gives sufficient condition for the mapping X.

Proposition 4.2. Let a file consist of two fields i, j whose sizes are less than the given number of devices M, and X be an injective function from N to Z_M. Then, FX distribution method with I-transformation for field i and X-transformation for field j is strict optimal for any range query in which field i is not range specified, if for any L consecutive values $J_1, \ldots, J_L$ in field j such that $L \le M/F_i$, $\{X(J_1), \ldots, X(J_L)\} = \{k_x F_i + c_x \mid x = 0, \ldots, L\text{-}1,$ and for all x, $k_x \in$ N, $k_x < M/F_i$, $0 \le c_x < F_i$, and $k_0 \ne \ldots \ne k_{L-1}\}$.

The proof for the proposition is straightforward from Lemma 3.2 (note that I-transformation function denotes the identity function). Based on Proposition 4.2, we describe an idea of how to find the mapping X in the following paragraph.

Let $F_j = 8$ and M = 16. The sizes of field i in which we are interested are 2, 4, and 8. Note that when the size of field i is greater than or equal to M, by Theorem 4.3 and Theorem 4.1 the distribution is strict optimal for any range query in which field i is not range specified. When $F_i = 8$, in order to satisfy the condition of the mapping X in Proposition 4.2, $X(0), X(1), \ldots, X(7)$ should be in interval [0, 8) and [8, 16), alternately. Here, "[" and "(" denote "closed" and "open", respectively. When $F_i = 4$, $X(0), X(1), \ldots, X(7)$ should be in interval [0, 4), [4, 8), [8, 12) and [12, 16) in turn. Since the sequence of intervals also has to satisfy the case when $F_i = 8$, the above sequence of intervals has to be reordered such as [0, 4), [8, 12), [4, 8) and [12, 16). When $F_i = 2$, $X(0), X(1), \ldots, X(7)$ should be in interval [0, 2), [2, 4), [4, 6), [6, 8), [8, 10), [10, 12), [12, 14), [14, 16) in turn. Similarly, we have to reorder the sequence of intervals such as [0, 2), [8, 10), [4, 6), [12, 14), [2, 4), [10, 12), [6, 8), [14, 16).

This idea can be stated as follows : When $(a_{m-1} \ldots a_0)_B$ is the binary notation of $J \in f_j$, where $m = \log_2 M$,

(1) For any consecutive two $J_1, J_2 \in f_j$, the most significant bit values of $X(J_1)$ and $X(J_2)$ have to be different

(2) For any consecutive four $J_1, J_2, J_3, J_4 \in f_j$, two most significant bits (i.e., left-most two bits) of $X(J_1), X(J_2), X(J_3)$ and $X(J_4)$ have to be all different.

(3) Thus, in general for any consecutive d values $J_1, \ldots, J_d \in f_j$, where d is a power of 2, the $\log_2 d$ most significant bits of $X(J_1), \ldots, X(J_d)$ have to be all different.

Based on these observations we propose the following field transformation functions.

Definition 4.7. Let $f_l = \{0, \ldots, F_l-1\}$ such that $|f_l|$ is a power of 2, and let $(a_s \cdots a_1 a_0)_B$ be the binary notation of $l \in f_l$. Then, $\mathbf{UR}^M : f_l \rightarrow Z_M$ is a function such that $\mathbf{UR}^M(l) = (a_0 a_1 \cdots a_{m-2} a_{m-1})_B$, where M is a power of 2 and $m = \log_2 M$.

Since we can add arbitrarily large number of 0's to the left of the given binary number of an integer l, function UR^M is always defined for any power of 2 integer M.

Definition 4.8. Let $f_l = \{0, \ldots, F_l-1\}$ such that $|f_l| < \mathrm{M}$, where $|f_l|$ and M are power of 2. Then, $\mathbf{UM}^{M, |f_l|} : f_l \rightarrow Z_M$ is a function such that $\mathbf{UM}^{M, |f_l|}(l) = \mathbf{UR}^M(l)$ [+] $(l \bmod d^{M, |f_l|})$, where $d^{M, |f_l|} = M/|f_l|$.

Example 4.5. When $f_1 = \{0, 1, 2, 3\}, f_2 = \{0, 1, 2, 3, 4, 5, 6, 7\}$ and M = 16,

(a) $UR^{16}(f_1) = \{0, 8, 4, 12\}$ and $UM^{16,4}(f_1) = \{0, 9, 6, 15\}$.

(b) $UR^{16}(f_2) = \{0, 8, 4, 12, 2, 10, 6, 14\}$, and $UM^{16,8}(f_2) = \{0, 9, 4, 13, 2, 11, 6, 15\}$.

Lemma 4.4. When there are only two fields i, j such that $|f_j|$ is less than the given number of devices M, functions UR^M and UM^{M,F_j} satisfy the condition of the function X in Proposition 4.2.

Proof: The proof of the lemma is straightforward from the definition of UR^M and UM^{M,F_j}. □

From the definition of the function *UR*, we can observe the following relations.

UR-property 1. Let $f_l = \{0, \ldots, F_l-1\}$ such that $|f_l| < M$. Then, for any $l \in f_l$, $UR^M(l) = UR^{F_l}(l) * M/F_l$.

The proof of UR-property 1 is straightforward from definition. Since $UR^{F_l}(f_l) = f_l$, UR-property 1 says that when the domain size of a function UR is less than the range size, the values of function *UR* are the multiples of the range size / the domain size.

UR-property 2. When $f_l = \{0, \ldots, F_l-1\}$ such that $F_l < M$, $UR^M(f_l) = U^{M,F_l}(f_l)$.

Since for any $l \in f_l$ such that $|f_l| < M$, $UR^M(l)$ is a multiple of M/F_l, and for any two different l_1 and $l_2 \in f_l$, $UR^M(l_1) \neq UR^M(l_2)$, it is easy to see UR-property 2. Thus, by UR-property 2, all the lemmas and theorems in [12, 20] which hold for *U*-transformation functions also hold for *UR*-transformation functions. Several useful characteristics of the function *UM* will be described in section 4.5.4 and 4.5.5.

Because of notational complexity we will use the following conventions : When the parameter M of a function *UR* denotes the given number of devices, we will leave out the parameter by default. When the parameter of a function *UR* does not mean the given number of devices, we will use the notation *BI* instead of *UR*. For example, UR^{F_l} in UR-property 1 will be denoted by BI^{F_l}. The parameter M and $|f_l|$ of function *UM* will be left out whenever there is no ambiguity.

Proposition 4.2 defines the class of function X such that FX distribution methods with $I(f_i)$ and $X(f_j)$ gives strict optimal distribution for any range query in which field i is not range specified. That proposition is only for the case when field i is *I*-transformed. In the following proposition we generalize Proposition 4.2 to include the case when other transformation functions are applied to field i.

Proposition 4.3. When a file consists of two fields i, j whose sizes are less than the given number of devices M, let X_i and X_j be transformation functions applied to field i

and j, respectively. Let $d_i = M/F_i$. Then, FX distribution methods with $X_i(f_i)$ and $X_j(f_j)$ are strict optimal for any range query in which field i is not range specified if

(1) $X_i(f_i)$ [+] $X_j(J_1)$ and $X_i(f_i)$ [+] $X_j(J_2)$ are disjoint for any $J_1, J_2 \in f_j$ such that J_1 [+] $J_2 < d_i$, and

(2) $X_i(f_i)$ [+] $X_j(J) = X_i(f_i)$ [+] $X_j(J[+]\alpha d_i)$ for any $J \in f_j$ and for any $\alpha \in$ N such that $J[+]\alpha d_i \leq F_j - 1$.

Proof: (1) and (2) in the proposition guarantee that for any $J \in f_j$, the sets $M(f_i, J)$, $M(f_i, J+1)$, . . . , $M(f_i, J+d_i-1)$ are disjoint if $J+d_i-1 \leq F_j-1$, and $M(f_i, J) = M(f_i, J+\alpha d_i)$ for any $\alpha \in$ N such that $J+\alpha d_i \leq F_j-1$. Thus, the proof follows. □

Proposition 4.3 gives general conditions which guarantee strict optimal distribution for a range query in which field i is not range specified. These conditions will be used frequently in proving theorems.

4.4.3. I and UR Field Transformation Functions

In this section we show that for any two fields i, j and the given number of devices M such that $F_j < M$, FX distribution methods are perfect optimal for type (0 - 1) range queries when I-transformation is applied to field i and UR-transformation is applied to field j.

Theorem 4.6. When there are only two fields i, j such that F_j is less than the given number of devices M, the FX distribution method with I-transformation for field i and UR-transformation for field j is (1) perfect optimal for type (0 - 1) range queries when $F_iF_j > M$, and (2) perfect optimal for type (0 - 2) range queries when $F_iF_j \leq M$.

Proof: (case 1) $F_iF_j > M$

When one field is range specified and the other field is specified as a single value, the proof follows from Theorem 4.4. When field i is range specified and field j is

unspecified, the proof follows from UR-property 2 (i.e., $UR(f_j) = U(f_j)$) and Lemma 3.8. When field j is range specified and field i is unspecified, the proof follows by Lemma 4.4 and Proposition 4.2. When there is no range specification, the proof follows by UR-property 2 and Theorem 3.4.

(case 2) $F_iF_j \leq M$

From UR-property 2 and Lemma 3.8, each device has at most one bucket, and therefore the proof follows. □

Example 4.6. Let $f_1 = \{0, 1, 2, 3\}$, $f_2 = \{0, 1, 2, 3\}$ and M = 8. Figure 4.7 shows the bucket distribution by the FX distribution method with $I(f_1)$ and $UR(f_2)$. Here, $UR(f_2) = \{0, 4, 2, 6\}$ and Device No = $T_M(I(J_1)\ [+]\ UR(J_2)), J_1 \in f_1, J_2 \in f_2$.

$I(f_1)$	$UR(f_2)$	Device No
000	000	0
000	100	4
000	010	2
000	110	6
001	000	1
001	100	5
001	010	3
001	110	7
010	000	2
010	100	6
010	010	0
010	110	4
011	000	3
011	100	7
011	010	1
011	110	5

Figure 4.7. FX Distribution with *I* And *UR* Transformation

We can verify that for any type 0 and type 1 range query the distribution of Figure 4.7 is strict optimal.

Before we describe the next section, it is worth considering the following question. Does there exist three field transformation functions X_1, X_2 and X_3 such that for any two fields whose sizes are less than M, FX distribution methods with any two of X_1, X_2, X_3 transformation functions are perfect optimal for type (0 - 1) range queries ? Unfornately, there do not exist such transformation functions because otherwise, it immediately contradicts Theorem 4.2. For example, let a file consist of three fields 1, 2 and 3 such that $F_1 = 4$, $F_2 = 4$ and $F_3 = 8$, and let M = 16. Suppose the above such functions X_1, X_2 and X_3 exist, and let us apply X_1, X_2 and X_3 transformation functions to field 1, 2 and 3, respectively. Then, it is easy to see that the distribution for this example file system should be strict optimal for any type 0 range query which has at most two unspecified fields, and strict optimal for any type 1 range query which has at most one unspecified field. However, by Theorem 4.2 we know that there does not exist such distribution for this file system. This implies there does not exist a transformation function Y such that FX distribution methods with *I* and Y-transformation as well as *UR* and Y-transformation is perfect optimal for type (0 - 1) range queries for every file with two fields. Thus, it is inevitable to have some restrictions for perfect optimal distribution for type (0 - 1) range queries when *I* and *UM*-transformation, and *UR* and *UM*-transformation are considered.

4.4.4. I and UM Field Transformation Functions

In this section we show that for any values of F_i, F_k and the given number of devices M such that $F_i \leq F_k < M$, FX distribution methods are perfect optimal for type (0 - 1) range queries when *I*-transformation is applied to field *i* and *UM*-transformation is applied to field *k*.

Lemma 4.5. When there are only two fields *i* and *k* whose sizes are less than the given number of devices M, the FX distribution method with *I*-transformation for field *i* and *UM*-transformation for field *k* is strict optimal for any range query in which field *i* is not

range specified.

Proof: This is a direct consequence of Lemma 4.4 and Proposition 4.2. □

Lemma 4.6. When the size of field k is less than the given number of devices M, let $d_k = M/F_k$. Then, for any two different nonnegative integers J_1 and J_2 such that J_1 and J_2 are within the same interval of size d_k (i.e., $J_1 \,[+]\, J_2 < d_k$), $UM(f_k) \,[+]\, J_1$ and $UM(f_k) \,[+]\, J_2$ are disjoint.

Proof: Let $J_1 = J'd_k \,[+]\, c_1$ and $J_2 = J'd_k \,[+]\, c_2$, where $J', c_1, c_2 \in$ N, and c_1 and c_2 are less than d_k. Assume the lemma is not true. Then, there exist two different $K_1, K_2 \in f_k$ such that

$$UR(K_1) \,[+]\, (K_1 \bmod d_k) \,[+]\, J'd_k \,[+]\, c_1 = UR(K_2) \,[+]\, (K_2 \bmod d_k) \,[+]\, J'd_k \,[+]\, c_2$$

Note that $UM(K_1) = UR(K_1) \,[+]\, (K_1 \bmod d_k)$ by definition. After removing $J'd_k$ from both sides of the above equality, we have

$$UR(K_1) \,[+]\, (K_1 \bmod d_k) \,[+]\, c_1 = UR(K_2) \,[+]\, (K_2 \bmod d_k) \,[+]\, c_2$$

This implies that $UR(K_1) \,[+]\, UR(K_2) = [(K_1 \bmod d_k) \,[+]\, c_1] \,[+]\, [(K_2 \bmod d_k) \,[+]\, c_2]$. Since $UR(K_1) \,[+]\, UR(K_2) > d_k$ by UR-property 1, and $(K_1 \bmod d_k) \,[+]\, c_1$, $(K_2 \bmod d_k) \,[+]\, c_2$ is less than d_k, there is no way the above equality can be satisfied. This is a contradiction. □

Lemma 4.7. When the size of field k is less than the given number of devices M, $UM(f_k) \,[+]\, UM(K) = UM(f_k)$ for any K $\in f_k$.

Proof: Since $UM(f_k) \,[+]\, UM(K)$ are a set of F_k different nonnegative integers, it is sufficient to show that for any two different K_1, K_2 in f_k, $(UM(K_1) \,[+]\, UM(K_2)) \in UM(f_k)$. Let $d_k = M/F_k$. Now,

$$UM(K_1) \,[+]\, UM(K_2) = [UR(K_1) \,[+]\, (K_1 \bmod d_k)] \,[+]\, [UR(K_2) \,[+]\, (K_2 \bmod d_k)]$$

$$= UR(K_1) \,[+]\, UR(K_2) \,[+]\, [(K_1 \,[+]\, K_2) \bmod d_k]$$

By UR-property 1, $UR(K_1)$ [+] $UR(K_2)$ is equal to $[BI^{F_k}(K_1)$ [+] $BI^{F_k}(K_2)]d_k$ (note that we defined BI^K to denote UR^K when the parameter K of function UR does not mean the given number of devices M). This implies that

$$UR(K_1) [+] UR(K_2) = BI^{F_k}(K_1 [+] K_2)d_k = UR(K_1 [+] K_2)$$

Thus, $UM(K_1) [+] UM(K_2) = UR(K_1 [+] K_2) [+] [(K_1 [+] K_2) \bmod d_k] = UM(K_1 [+] K_2)$. Since $K_1 [+] K_2 \in f_k$ by Lemma 3.2, the proof follows. □

Lemma 4.8. Let the size of field k be less than the given number of devices M. Then for any $K \in f_k$, $UM(f_k) [+] UR(K) = UM(f_k) [+] (K \bmod d_k)$, where $d_k = M/F_k$.

Proof: Let $K \in f_k$. Then,

$$UM(f_k) [+] UR(K) = UM(f_k) [+] UR(K) [+] (K \bmod d_k) [+] (K \bmod d_k)$$

$$= UM(f_k) [+] UM(K) [+] (K \bmod d_k)$$

Since $UM(f_k) [+] UM(K) = UM(f_k)$ by Lemma 4.7, the proof follows. □

Lemma 4.9. When there are only two fields i and k whose sizes are less than the given number of devices M and $F_i \le F_k$, the FX distribution method with I-transformation for field i and UM-transformation for filed k is strict optimal for any range query in which field k is not range specified.

Proof: Let $d_k = M/F_k$. By Lemma 4.6, for any two different K_1 and K_2 which are in the same interval of size d_k, $UM(f_k) [+] K_1$ and $UM(f_k) [+] K_2$ are disjoint. Thus, by Proposition 4.3 it is sufficient to show that for any $J \in f_i$, $UM(f_k) [+] J = UM(f_k) [+] (J [+] \alpha d_k)$, where α is any nonnegative integer such that $\alpha d_k < F_i$. Now,

$$UM(f_k) [+] \alpha d_k = UM(f_k) [+] UR(BI^{F_k}(\alpha))$$

$$= UM(f_k) [+] (BI^{F_k}(\alpha) \bmod d_k)$$

The first equality holds because $UR(BI^{F_k}(\alpha)) = BI^{F_k}(BI^{F_k}(\alpha))d_k$ by UR-property 1, and $BI^{F_k}(BI^{F_k}(\alpha)) = \alpha$. The second equality holds by Lemma 4.8. Since $\alpha < F_k/d_k$

(because $F_i \le F_k$), this implies that $BI^{F_k}(\alpha)$ is a multiple of d_k for all $\alpha = 0, \ldots, F_i/d_k - 1$. Thus, $BI^{F_k}(\alpha)\ mod\ d_k = 0$ and therefore $UM(f_k)\ [+]\ \alpha d_k = UM(f_k)$ for all $\alpha = 0, \ldots, F_i/d_k - 1$. This completes the proof. □

Theorem 4.7. When there are only two fields i and k whose sizes are less than the given number of devices M, the FX distribution method with I-transformation for field i and UM-transformaton for field k is (1) strict optimal for any range query in which field i is not range specified if $F_i > F_k$, and (2) perfect optimal for type (0 - 1) range queries if $F_i \le F_k$, and (3) perfect optimal for type (0 - 2) range queries if $F_iF_k \le M$.

Proof: For (1) and (2), the theorem is a direct consequence of Lemma 4.5 and Lemma 4.9. When $F_iF_k \le M$, since $F_i - 1 < M/F_k$, the theorem is a direct consequence of Lemma 4.6. □

Example 4.7. Let $f_1 = \{0, 1, 2, 3\}$, $f_2 = \{0, 1, 2, 3\}$ and M = 8. Figure 4.8 shows the bucket distribution by FX distribution method with $I(f_1)$ and $UM(f_2)$. Here, $UM(f_2) = \{0, 5, 2, 7\}$ and Device No = $T_M(I(J_1)\ [+]\ UM(J_2)), J_1 \in f_1, J_2 \in f_2$.

We can verify that for any type 0 and type 1 range query the distribution in the figure is strict optimal.

4.4.5. UR and UM Field Transformation Functions

In this section we discuss the cases of optimal distribution when UR and UM field transformation functions are used.

Lemma 4.10. Let the size of field k be less than the given number of devices M. Then, for any $K \in f_k$, $K\ mod\ d = UM(K)\ mod\ d$, where d is a power of 2 which is less than or equal to M/F_k.

Proof: Let $d_k = M/F_k$.

$$UM(K)\ mod\ d = [UR(K)\ [+]\ (K\ mod\ d_k)]\ mod\ d$$

$$= [UR(K)\ mod\ d]\ [+]\ [(K\ mod\ d_k)\ mod\ d]$$

$I(f_1)$	$UM(f_2)$	Device No
000	000	0
000	101	5
000	010	2
000	111	7
001	000	1
001	101	4
001	010	3
001	111	6
010	000	2
010	101	7
010	010	0
010	111	5
011	000	3
011	101	6
011	010	1
011	111	4

Figure 4.8. FX Distribution with *I* And *UM* Transformation

Since $UR(K)$ is a multiple of d_k by UR-property 1, and d_k is a multiple of d, $UR(K)\ mod\ d = 0$ and $(K\ mod\ d_k)\ mod\ d = K\ mod\ d$. Thus, the proof follows. □

Lemma 4.11. When there are only two fields j and k whose sizes are less than the given number of devices M, the FX distribution method with *UR*-transformation for field j and *UM*-transformation for field k is strict optimal for any range query in which field j is not range specified, if (1) $F_j \geq F_k$ or (2) $F_jF_k \leq$ M.

Proof: When one field is range specified and the other field is specified as a single value, the proof follows from Theorem 4.4. Thus, we have only to consider the case when field j is unspecified, and field k is range specified or unspecified.

(case 1) $F_j \geq F_k$

Let $d_j = M/F_j$ and $d_k = M/F_k$. Then, $d_j \leq d_k$. By UR-property 2, $UR(f_j) = U(f_j)$, and by Lemma 4.10, $UM(K)\ mod\ d_k = K\ mod\ d_k$ for any K in f_k. Since $d_j \leq d_k$, by Lemma

3.8 the proof follows.

(case 2) $F_jF_k \leq M$

This is a direct consequence Lemma 4.10 and Lemma 3.8. □

Lemma 4.12. When there are only two fields j and k whose sizes are less than the given number of devices M, the FX distribution method with *UR*-transformation for field j and *UM*-transformation for field k is strict optimal for any range query in which field k is not range specified, if (1) $F_j \leq F_k$ or (2) $F_jF_k \leq M$.

Proof: (case 1) $F_j \leq F_k$

Let $d_k = M/F_k$. By Lemma 4.6, we know for any two different nonnegative integers J_1 and J_2 such that J_1 and J_2 are within the same interval of size d_k (i.e., J_1 [+] $J_2 < d_k$), $UM(f_k)$ [+] J_1 and $UM(f_k)$ [+] J_2 are disjoint. Thus, by Proposition 4.3 it is sufficient to show that for any $J \in f_j$, $UM(f_k)$ [+] $UR(J) = UM(f_k)$ [+] $UR(J$ [+] $\alpha d_k)$, where α is any nonnegative integer such that $\alpha d_k < F_j$. Now, since J is also an element of f_k because $F_j \leq F_k$, by Lemma 4.8 $UM(f_k)$ [+] $UR(J) = UM(f_k)$ [+] $(J \ mod \ d_k)$ for any $J \in f_j$. This implies that

$$UM(f_k) \text{ [+] } UR(J \text{ [+] } \alpha d_k) = UM(f_k) \text{ [+] } [(J \text{ [+] } \alpha d_k) \ mod \ d_k]$$

$$= UM(f_k) \text{ [+] } (J \ mod \ d_k)$$

Note that J [+] αd_k is also an element of f_k. Thus, by Proposition 4.3 the proof follows.

(case 2) $F_jF_k \leq M$

This is a direct consequence of Lemma 4.10 and Lemma 3.8. □

Theorem 4.8. When there are only two fields j and k whose sizes are less than the given number of devices M, the FX distribution method with *UR*-transformation for field j and *UM*-transformation for field k is (1) strict optimal for any range query in which field j is not range specified if $F_j \geq F_k$, and (2) strict optimal for any range query in which field k is not range specified if $F_j \leq F_k$, and (3) perfect optimal for type (0 - 1) range queries if $F_j = F_k$, and (4) perfect optimal for type (0 - 2) range queries if

$F_j F_k \leq M$.

Proof: The theorem is a direct consequence of Lemma 4.11, Lemma 4.12 and Lemma 3.8. □

Example 4.8. Let $f_1 = \{0, 1, 2, 3\}$, $f_2 = \{0, 1, 2, 3\}$ and M = 8. Figure 4.9 shows the bucket distribution by FX distribution with $UR(f_1)$ and $UM(f_2)$. Here, $UR(f_1) = \{0, 4, 2, 6\}$, $UM(f_2) = \{0, 5, 2, 7\}$ and Device No = $T_M(UR(J_1)$ [+] $UM(J_2))$, $J_1 \in f_1, J_2 \in f_2$.

$UR(f_1)$	$UM(f_2)$	Device No
000	000	0
000	101	5
000	010	2
000	111	7
100	000	4
100	101	1
100	010	6
100	111	3
010	000	2
010	101	7
010	010	0
010	111	5
110	000	6
110	101	3
110	010	4
110	111	1

Figure 4.9. FX Distribution with *UR* And *UM* Transformation

We can verify that the distribution is perfect optimal for type (0 - 1) range queries.

4.4.6. I, UR and UM Field Transformation Functions

In this section we discuss the cases of optimal disribution when *I*, *UR* and *UM*-transformation functions are used.

Theorem 4.9. When there are only three fields i, j and k such that $F_i \le F_j \le F_k < M$, FX distribution methods can be always (1) perfect optimal for type 0 range queries, and (2) strict optimal for any type 1 range query which has at most one unspecified field if $F_j = F_k$, or $F_i F_k \le$ M.

Proof: (1) is true by Corollary 3.4. Thus, let us consider only the case (2). When $F_i \le F_j = F_k$, let field i be I-transformed, field j be UR-transformed and field k be UM-transformed. When $F_i F_k \le M$, let field i be UR-transformed, field j be I-transformed and field k be UM-transformed. Then, for both cases, the theorem is a direct consequence of Theorem 4.6, Theorem 4.7 and Theorem 4.8. □

Note that by Theorem 4.2 there does not exist a data distribution method which guarantees strict optimal distribution for any range query in Theorem 4.9 for every file with three fields.

We have shown through lemmas and theorems that FX distribution methods along with various combinations of field transformation functions give optimal distribution for many types of range queries. Here, it should be emphasized that the scope of optimality is increased significantly by these field transformation techniques along with Proposition 4.1. This is because by Proposition 4.1 optimal distribution for a subset of the fields guarantees strict optimal distribution for many range queries in which those fields are range specified or unspecified.

CHAPTER 5

NODE PARTITIONING SCHEMES FOR B-TREES

5.1. Introduction

In chapter 3 and 4 we have described data distribution strategies for multikey search queries which access a set of records for each query. These are type A1 applications based on the classification in section 1.4. In this chapter we will investigate data distribution strategies for parallel processing of type A2 applications, i.e., the database query which accesses a record based on primary key. The object is to enhance concurrency by parallel processing of tree type index structure for the database stored in the external storage. Parallel processing strategy for key based access main memory databases can be found in [Cev88].

B-trees are the most commonly used tree type index structure for external databases [Bay72, Com79]. For the database stored in the secondary storage, the height of an index tree is the most important parameter in data retrieval time. One approach to reduce the height of a B-tree is to partition a file horizontally, and construct a B-tree for each subset of the partition. These B-trees are searched in parallel. When the database size is D and is partitioned into p subsets, the height of each B-tree is approximately $\log_f(D/p)$, where f is the fan-out of an index node. Another approach to reduce the height of a B-tree is to use a large node B-tree for the whole file. When the size of index nodes is increased p times, the height of a B-tree becomes about $\log_{pf}D$. The second method gives smaller height because $D > pf$ for most practical cases. However, this approach alone may not improve the overall performance because large nodes result in long block transfer time and more main memory processing time.

We propose a node partitioning scheme for large node B-trees for parallel processing. In the proposed scheme, each node is partitioned into multiple subnodes to be distributed for parallel processing. We also propose a new approach to process these subnodes. We will call the proposed B-tree structure Partitioned Node B-trees (PNB-trees), while the standard B-trees will be called conventional B-trees. With respect to the two stage processing model in Figure 2.2, PNB-trees are based on partitioned node B-tree data construction, and random or object specific data distribution.

The main results presented in this chapter are that the parallel processing of PNB-trees reduces access time, increases throughput and minimizes the frequency of tree restructuring. The basic structure of PNB-trees is based on B-trees. However, we use a different approach to process and search PNB-trees. The PNB-tree approach exploits parallel scanning by distributing multiple subnodes of a B-tree node among parallel disks.

The rest of this chapter is organized as follows. In section 5.2 we present the basic structure of PNB-trees, and the search and update algorithms. The hardware environment for the PNB-tree construction is also discussed in this section. The important parameters of the PNB-tree are described in section 5.3. Section 5.4 presents a performance comparison between PNB-trees and conventional B-trees. Finally, PNB-tree construction for various disk technology is discussed in section 5.5.

5.2. The PNB-tree

PNB-trees are constructed on synchronized disks where read/write heads of all the disks are located at the same position. Synchronizing disks for disk interleaving was proposed in [Kim86] where multiple disks are interleaved like main memory interleaving. It has been shown in that paper that the response time improves considerably by using disk interleaving techniques when block size is large. This is because synchronized disk interleaving increases data transfer rate significantly. In PNB-trees,

synchronized disks are used to exploit searching in parallel.

The PNB-tree is a B-tree with each node partitioned into s subnodes, and all the subnodes of a node are stored on s different disks. The format of an index node is $(\phi,P_0)(K_1,P_1)(K_2,P_2) \cdots (K_m,P_m)$ where K_i is the key value and P_i is the address of the child subnode in each disk. This index node is partitioned into s subnodes, and all the subnodes of a node are accessed by the same address pointer. All the subnodes have the same format except the first one, where the key value in the first record is omitted. The leaf nodes which contain the data are also partitioned into s subnodes. However, PNB-trees differ from conventional B-trees in that the key values within a node in PNB-trees are not required to be in sorted order. This needs extra computation to determine the child address, but the amount of data movement between disks is reduced considerably because a subnode does not overflow. The rule of locating a data record is defined recursively as follows :

> Let R = $\{(\phi , P_0), (K_1 , P_1), (K_2 , P_2), \cdots , (K_m , P_m)\}$ be a current index node. Then the data record with a key value K is in the descendant node pointed by P_y, if $(K_y , P_y) \in$ R - $\{(\phi ,P_0)\}$ and $(K - K_y)$ is the minimum nonnegative value among all $(K - K_i)$, $i = 1,...,m$. If there is no nonnegative $(K - K_i)$, $i = 1,...,m$, the desired record is in the descendant node pointed by P_0.

An example of a PNB-tree with a set of key values is given in Figure 5.1. It shows that each node is partitioned into two subnodes being stored on disks D_1 and D_2. Note that the values in a node are not kept in sorted order. When the key value e.g., 25 needs to be searched, we first compute the differences $25 - K_y$ for all the key values K_y in the root node. These values are <-7, 4, -66, -46>. By following the pointer associated with K_y such that $25 - K_y = 4$, the node <21, 25, 23, 29> is obtained. Here it is assumed that all data values are stored in leaf nodes.

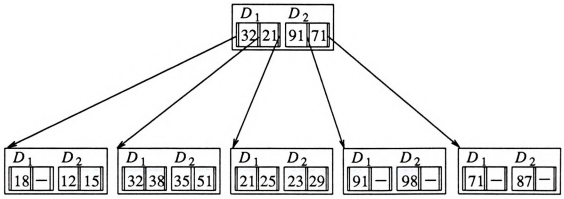


Figure 5.1. Partitioned Node B-tree

The reason for the use of unsorted nodes is to minimize the inter-device data movement. Suppose that a node is in sorted order, and is partitioned and stored as above. Whenever a subnode overflows, data movement between disks is required. However, this does not happen in unsorted node construction because inserted records can be placed in any subnode of a node.

5.2.1. The PNB-tree Operations

(1) LOOKUP

The lookup of a record starts from the root node and continues until the leaf. All the subnodes of a node are searched in parallel. Let P be the address of the node currently being accessed. Initially, P is the address of the root. The lookup procedure consists of two phases. At first, (K, P) is broadcast to all the disks, where K is the desired key value. If P is the address of a leaf node, find the record and stop. If not, each disk finds an index record (K_i, P_i) in its subnode such that $(K - K_i)$ is the minimum nonnegative value. In the second phase, find the minimum of these minimum nonnegative values at each disk. If no nonnegative value is found at any disk, repeat the first and the second phase by replacing P with P_0. Otherwise, repeat the same by

replacing P with P_j, where (K_j , P_j) is an index record and K_j is the overall minimum nonnegative value.

(2) INSERTION

To insert a record, apply the lookup procedure to find the desired leaf node. We then insert the record into any place within the node, if it has empty room. If the node is already full, it is sorted and then split into two nodes. The effect of node splitting on its parent node is handled recursively in the same way as in standard B-trees. Figure 5.2 shows the PNB-tree configuration after the key value 57 has been inserted into the tree of Figure 5.1.

(3) DELETION

The node underflow in PNB_trees is handled the same way as in standard B-trees. PNB-trees have an advantage over B-trees because the holes created by deletion can be easily filled by any records inserted. This is because the values within a node need not be kept in sorted order. In conventional B-trees the holes can also be used but only at the cost of sorting the node each time. Figure 5.3 shows the PNB-tree configuration when key value 71 is deleted from the tree of Figure 5.2.

5.2.2. Parallel Disks Configuration for PNB-trees

For each disk we assume a simple microprocessor with a few thousand bytes of memory. All the subnodes of a node are processed locally by these processors. Only the matched record is sent to the host processor. For the second phase of a lookup procedure, we use a minimum finding hardware module to which all the disks are directly connected. This module is a simple extension of a multiple input comparator. As soon as each disk finds an index record (K_i,P_i) such that "the desired key value $- K_i$" is minimum nonnegative value of that disk, it sends that index record to this minimum finding hardware. If a disk does not find a nonnegative value, it sends (–1, *null*). The

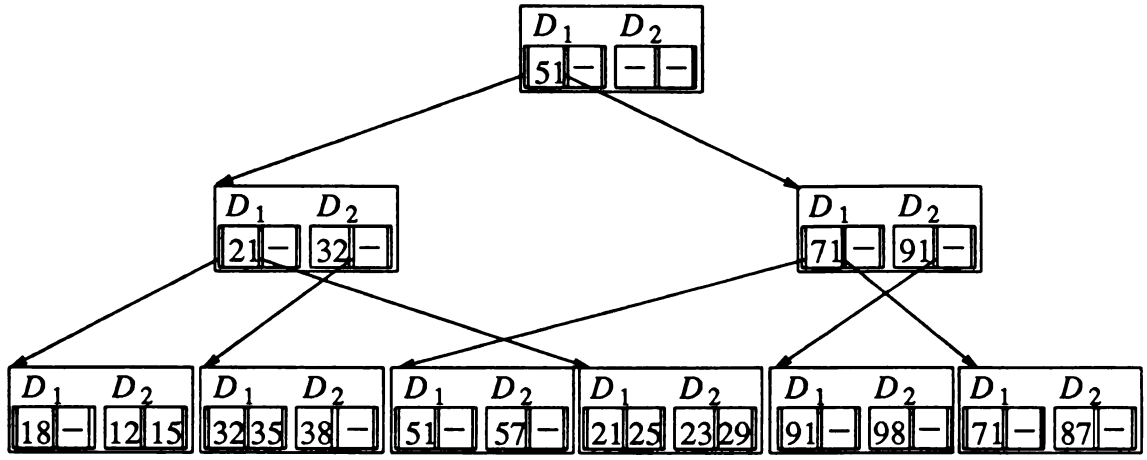


Figure 5.2. PNB-tree After Insertion of Key Value 57

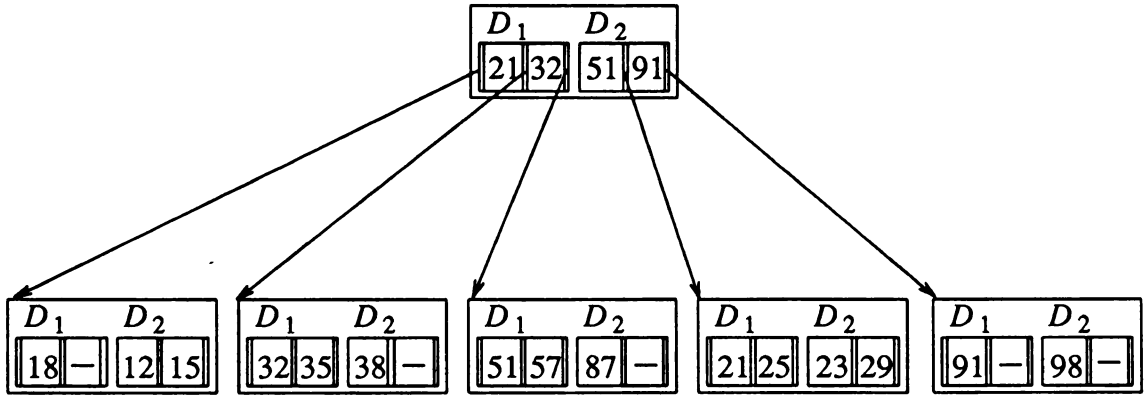


Figure 5.3. PNB-tree After Deletion of Key Value 71

minimum finding hardware determines the overall minimum nonnegative value and then broadcasts the corresponding address of the child node to all the disks. When all the disks reply with (−1, *null*), the address of a child node is the first pointer in the first subnode of a node. Since the communication involved is very simple and communication distance is short, we expect that the time for these operations is negligible compared to disk access time.

5.3. Motivations of PNB-trees

The motivations of PNB-trees are to reduce the tree height and to reduce the frequency of tree restructuring. This is because the height of index trees and the frequency of tree restructuring are very important parameters for the performance of external databases.

5.3.1. Compressed Height

Let s be the number of disks, n be the number of records, r be the size of a data record, and k be the key size. Let the block size be b bytes and the pointer size be p bytes, where a block is a unit of data transfer. This block corresponds to a subnode in PNB-trees and a node in conventional B-trees. Let h be the height of a PNB-tree, where h includes the level of data nodes. To compute the worst case height we assume every block to be half-full except the root node. Then, the maximum number of leaf nodes is approximately

$$\frac{s^h b}{k+p}\left[\frac{b}{2(k+p)}\right]^{h-2}$$

Since we need about $2nr/b$ data blocks (leaf nodes), the relation

$$\frac{s^h b}{k+p}\left[\frac{b}{2(k+p)}\right]^{h-2} \geq \frac{2nr}{b}$$

should be satisfied [Knu73]. For all the examples in this chapter we will use $r = 200$

bytes, $k = 30$ bytes and $p = 4$ bytes. Then, $b \geq 2500$ if $h = 3$, $s = 4$ and $n = 1$ M. Therefore, 3 Kbyte block size is enough for making 2 level indices in PNB-trees. Even for large files with 60 M records, the block size of 5 Kbyte guarantees 2 level indices in PNB-trees when eight disks are used. On the other hand, 4 level indices are needed for this file when the conventional B-tree of 5 Kbyte block size is used. To guarantee 2 level indices in conventional B-trees we need to use the block size of about 48 Kbyte. In this case the block transfer time is quite significant.

The node size of PNB-trees is large, but I/O and computation time improves significantly because the large node is split into smaller subnodes and these subnodes are processed in parallel. Since the height of a B-tree decreases with increasing node size, the number of disk accesses in PNB-trees is minimized. Furthermore, the block transfer time is reduced because only the subnode contributes to the block transfer time. Note that the height can also be reduced in conventional B-trees by making the node size large. But this results in a long block transfer time and more main memory processing time.

5.3.2. Reduced Frequency of Tree Restructuring

The B-tree restructuring due to node overflow is expensive. One way to reduce the frequency of tree restructuring is to use a large node. When the node size increases k times, the frequency of tree restructuring due to node overflow reduces by a factor of $1/ck$, where c is slightly larger than one. This is because the number of nodes in a B-tree decreases almost linearly with node sizes. The detailed analysis for computing the average number of nodes in a B-tree is given in [Yao79].

5.4. Performance Comparison

In this section we compare the response time of PNB-trees with conventional B-trees which are stored on parallel disks.

5.4.1. Performance Models

For conventional B-trees each data request is directed to the disk which contains the corresponding B-tree. Thus, a request queue is formed in front of each disk. In PNB-trees all data requests form one queue.

Let X be a random variable for the *disk service time* of a single disk. Then, X = S(seek time) + L(rotational latency) + T(block transfer time). For conventional B-trees on parallel disks Rotational Positioning Sensing (RPS) miss delay should be added for disk access time [Kim86]. RPS miss delay happens due to the channel contention by concurrent disk I/Os. Let X_i be the disk service time for disk i in conventional parallel disks. Then, X_i = S + L + T + RPS_i. We define the *disk response time* to be the disk service time plus the queue waiting time for the disk service. The queue waiting time is computed as follows. The arrival process for disk access requests is assumed to be Poisson process with mean R. L is uniformly distributed with mean one half of one disk rotation time, S is assumed to be exponentially distributed and the number of RPS_i misses is assumed to be geometrically distributed [Kim86]. Since the distributions of S, L ,T and RPS_i are known, the mean and the variance of X_i can be computed. We choose M/G/1 queueing model to compute average queue waiting time. In M/G/1 queueing model, the queue waiting time for disk i, denoted W_i, is given by

$$\frac{R_i(Var(X_i)+E[X_i]^2)}{2(1-R_i*E[X_i])}$$

where R_i is the request rate for disk i [Ross85]. Thus, the average disk response time for disk i is $E[X_i] + W_i$. Let Z_j be the disk response time for a j-th level node of a B-tree. We define the *data response time* Z to be $Z_1 + Z_2 + \cdots + Z_h$, where h is the height of the B-tree. Thus, the average data response time $E[Z] = hE[Z_j]$. Here, for all $j = 1, \ldots, h$,

$$E[Z_j] = \frac{1}{s}\sum_{i=1}^{s}(E[X_i] + W_i)$$

where s is the number of disks. For the PNB-tree model,

$$E[Z_j] = E[X] + \frac{R(Var(X)+E[X]^2)}{2(1-R*E[X])}$$

for all $j = 1, \ldots, h$, where R is equal to $\sum_{i=1}^{s} R_i$. Note that there is only one queue and there is no RPS miss delay in the PNB-tree model.

As described in section 5.2, PNB-trees use two phase processing to locate a record. In the first phase, computation time is overlapped with data transfer time because processing can be done concurrently with the data transfer between a disk and a local buffer. For the second phase, we assume that it takes less than one millisecond to find the overall minimum value.

5.4.2. Average Data Retrieval Time

In this section we use IBM 3380 disk parameters which are E[S] = 7.2 ms, E[L] = 8.3 ms and block transfer rate = 3 Mbyte/sec. We will compare the performance of conventional B-trees on parallel disks with PNB-trees on synchronized disks. As typical examples, four disks and eight disks are used. In conventional parallel disks, data requests are usually skewed for some disks. The following skewed data request patterns are used in the performance experiments.

1)

D1	*D2*	*D3*	*D4*
.458	*.371*	*.153*	*.018*

2)

D1	*D2*	*D3*	*D4*	*D5*	*D6*	*D7*	*D8*
.388	*.225*	*.153*	*.102999*	*.000001*	*.010*	*.054*	*.068*

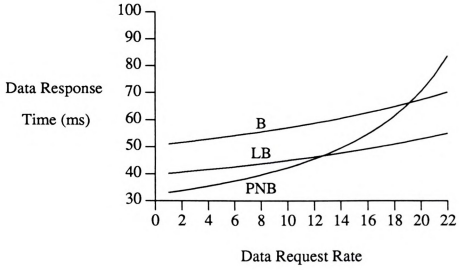
Here, *Di* represents disk *i* and the number below *Di* represents the probability that some request is a disk *i* request. Note that there is no skewed effect in the PNB-tree model because there is only one queue. Let the number of records be 1 M and all other parameters be the same as in section 5.3.1. When 4 disks are used, the data response time for conventional B-trees is 70.07 ms while data response time for PNB-tree is 54.09 ms.

We have used 4 Kbyte block size with a data request rate of 2 per second. When 8 disks are used, block size is 2 Kbyte and the data request rate is 2 per second, the data response time of conventional B-trees is 77.85 ms and that for the PNB-trees is 51.85 ms. The root node is assumed to be resident in main memory for all response time computation. The height of the B-tree and the PNB-tree is computed with the assumption that every node is half-full. We see that the use of small block size for conventional B-trees may increase the data response time due to the increased height. In PNB-trees we can have smaller block size with the same height by using more disks. Thus, the PNB-tree takes advantage of less block transfer time. In conventional disk systems with small block size, the block transfer time is negligible but the time due to increased disk accesses is significant.

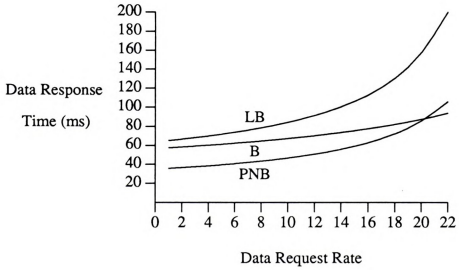
Many parameters need to be considered in comparing the response time of conventional B-trees on parallel disks and PNB-trees on synchronized disks. This is because the height of B-trees depends on the block size, the number of records, record size and key field size. We will use the same values for these parameters as in section 5.3.1 for the following experiments.

We compare three different B-tree processing models. The first is conventional B-trees with height 4. The second is conventional B-trees with height 3, and the third is PNB-trees with height 3. The second model uses an increased block size to reduce the height of the conventional B-tree. In the figures B, LB and PNB denote the response time of the first, second and third model, respectively.

Figure 5.4 shows response time for various data request rate when IBM 3380 disk parameters are used. In Figure 5.4.(a), we used a database containing 1 M records with eight parallel disks. The block sizes for conventional B-trees are 4 Kbyte and 14 Kbyte. The block size for the PNB-trees is 2 Kbyte. In the figure PNB-trees are the most efficient, until 12 data requests/sec. Conventional B-trees of height 3 by using a large block size is always better than the conventional B-trees of height 4. But for very large



(a) 1 M Records



(b) 60 M Records

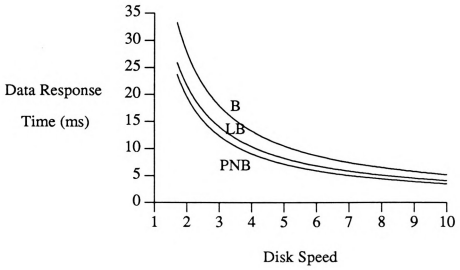
Figure 5.4. Data Response Time with Various Data Request Rates

files this is not the case. Figure 5.4.(b) shows the case for 60 M records with eight disks. Here, we use 10 Kbyte and 48 Kbyte block sizes for the conventional B-trees, and 6 Kbyte block size for the PNB-trees. The PNB-trees are the most efficient, until 21 data requests/sec. Using the large block size in the conventional B-trees turns out to be the worst. This is because for 48 Kbyte block size the block transfer time is comparable to one disk access time. Furthermore, it suffers severe RPS miss delay due to the long block transfer time. On the other hand, PNB-trees perform better than conventional B-trees for both small and large block sizes.

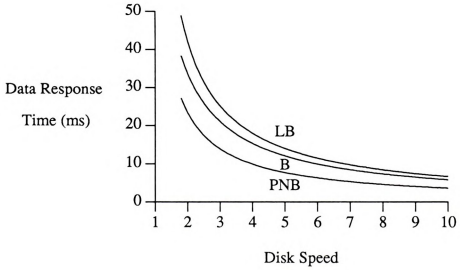
Figure 5.5 shows the response time comparison for various disk speeds, when data request rate is 20/sec. Disk speed i in these figures denotes $1/i \times$ "average disk access time of IBM 3380 disk". Figure 5.5.(a) and 5.5.(b) show the results for 4 parallel disks with 1 M records and for 8 parallel disks with 60M records, respectively. In Figure 5.5.(a) 4 Kbyte and 14 Kbyte block sizes are used for conventional B-trees, and 4 Kbyte block size is used in PNB-trees. In Figure 5.5.(b) we used 10 Kbyte and 48 Kbyte block sizes for conventional B-trees and 6 Kbyte block size for PNB-trees. In this figure we also see that PNB-trees perform better than conventional B-trees for both small and large block sizes.

The *saturation request rate* is defined to be a data request rate which makes the queue length infinite. The PNB-tree achieves a larger saturation request rate than the conventional B-tree in the worst case, where every request goes to the same disk. Note that in PNB-trees the worst case is the same as the average case. When the number of records is 1 M and 8 disks are used with 4 Kbyte block size, the PNB-tree is saturated at 21 data requests/sec, but the conventional B-tree is saturated at 14 data requests/sec. Therefore, PNB-trees can handle larger data request rate in the worst case.

One problem of PNB-tree organization is an increased queue length. This is because all disk requests go to the same queue. For normal data request rate, the queue length is usually very small(less than 0.1). But for high data request rate(e.g., more



(a) 1 M Records



(b) 60 M Records

Figure 5.5. Data Response Time with Various Disk Speeds

than 15 data requests per second which are more than 45 disk access requests per second, if the height of a B-tree is 3), the queue waiting time is no longer negligible and the PNB-tree suffers from long queue waiting time.

In the next section the range of data request rates until which the benefit of PNB-trees is larger than the increased queue waiting time will be given. Note that the range of data request rates resulting in long queue waiting time is quite dependent on the disk speed.

5.4.3. Threshold Points

We define the *threshold point* to be the maximum data request rate until when the response time of the PNB-tree is smaller than the conventional B-tree. Figure 5.6 shows threshold points for various disk speeds. 1 M records and 4 parallel disks are used.

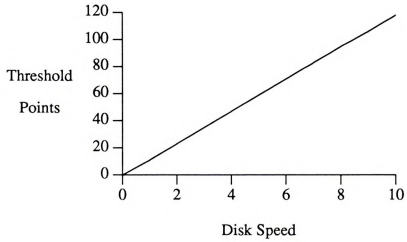


Figure 5.6. Threshold Points

In the figure we can observe that threshold limit increases almost linearly with the disk speed.

5.4.4. Update Performance

As we discussed in section 5.3.2, PNB-trees have an advantage of reduced number of tree restructuring for insertion/deletion operation. Figure 5.7 shows the total number of tree restructuring and Figure 5.8 shows the total amount of time for creating files of different file sizes for conventional B-trees and PNB-trees. Here, eight IBM 3380 disks with the conventional B-tree of 4 Kbyte and 32 Kbyte block size, and the PNB-tree of 4 Kbyte block size are used. In both figures, B denotes the conventional B-tree of 4 Kbyte block size, LB denotes the conventional B-tree of 32 Kbyte and PNB denotes the PNB-tree of 4 Kbyte block size. In Figure 5.7 the conventional B-tree of 32 Kbyte block size has almost the same shape as that of the PNB-tree. When the insertion/deletion request rate is high, the advantage of the reduced number of tree restructuring for the PNB-tree is quite significant.

5.5. Other Strategies for The PNB-tree Organization

In synchronized disks, discussed so far, we use one pointer to locate all the subnodes of a node. When asynchronous disks are used for PNB-trees, multiple pointers are needed to locate multiple subnodes. Note that asynchronous disks do not use synchronizing disk-arm. In spite of a little bit extra storage to maintain multiple pointers, this approach gives much better overall storage utilization. This is because the records can be stored at any place within a node since key values need not be kept in sorted order. Therefore, we can store the data in a compact form with variable number of subnodes for each node. For this implementation, all the tree operation algorithms are the same as those in section 5.2. One problem of this approach is the increased disk access time because the disk access time is determined by the worst case disk position (note that the second phase of lookup operation requires responses of all disks).

Another approach is to use synchronized disks and keep the key values ordered within a node. By keeping ordered key values in a node we can eliminate associated

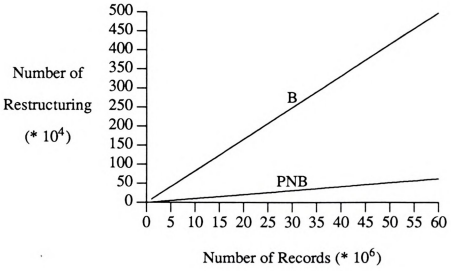


Figure 5.7. Tree Restructuring Cost

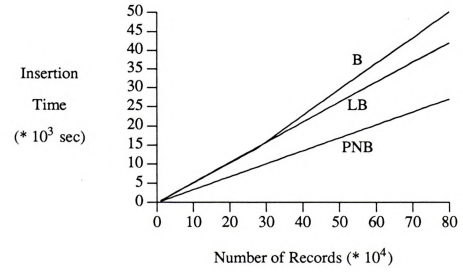


Figure 5.8. Insertion Time

minimum finding hardware for each disk. This approach has the problem of increased number of tree restructuring due to subnode overflow. Note that restructuring operation is expensive.

We can also use asynchronous disks and keep the key values ordered within a node. The disk access time for a successful search does not increase in this method. This is because for a successful search the decision can be made immediately in one disk without collecting responses from all the disks. Therefore, for a successful search the disk access time is the same as that of synchronized disks. However, this approach requires multiple pointers and has the same problem as synchronized disks with ordered key values.

CHAPTER 6

CONCLUSIONS

We have investigated data distribution strategies for various database applications. We have proposed a general database parallel processing model and a two stage database parallel processing model which can be used as a framework for more specific implementation. The general database parallel processing model shows important issues of parallel database systems. The two stage database parallel processing model is derived from the general database parallel processing model and is suitable for many database applications. The objective of these models is to facilitate the design of parallel processing database systems. We have applied these abstract models to three specific database applications. These are partial match queries, multiattribute range queries and key based accesses on B-tree type index structures.

We have investigated concurrent data accesses for partial match queries. Much research has been done on designing efficient multikey hashing schemes for partial match retrieval type applications. We have focused on optimal bucket distribution in multikey hashing to achieve maximum access concurrency for partial match queries. We have presented FX distribution methods which are based on exclusive-or operation. Field transformation techniques have been used to increase the scope of optimality in FX distribution methods. Performance of FX distribution methods has been compared with those of the other existing methods for typical file systems. We have shown that FX distribution methods give higher probability of strict optimality than the existing methods. We have also shown that the query response time of FX distribution methods is better than that of the others.

Concurrent data accesses on multiattribute range queries has also been investigated. Though much work has been done in designing file systems for multiattribute range queries to reduce the number of bucket accesses, appropriate data distribution among the access nodes has not been considered for these applications to enhance access concurrency. There are several proposals for data distribution methods for multikey accesses. However, they did not consider range specification in a query. We have investigated optimal data distribution for multiattribute range queries in parallel processing file systems. Since perfect optimal distribution is the most desirable, the existence of perfect optimal distribution has been investigated. We have shown for various types of range queries that there are inherent limitations to achieve optimal distribution. The results show that optimal distribution does not exist in many cases even for files with two fields. We have given sufficient conditions for the nonexistence of perfect optimal distribution for certain types of range queries. We have also developed data distribution methods for several useful multiattribute range queries. Sufficient conditions for optimal distribution by these proposed methods have been given. It has been shown that the proposed data distribution methods are perfect optimal for certain types of multiattribute range queries, and strict optimal for a large class of multiattribute range queries.

We have proposed data distribution strategies for B-tree nodes to improve the response time of file accesses. This approach is based on the partitioned node B-tree called the PNB-tree. We compared the performance of PNB-trees with conventional B-trees on parallel disks. PNB-trees reduce the height of B-trees and exploit intra-node parallel processing. We have shown that the height reduction of the B-tree along with parallel processing within a node gives better performance than parallel processing of conventional B-trees on parallel disks. We have also shown that the frequency of tree restructuring due to node overflow is considerably reduced in PNB-trees.

This thesis investigated optimal data distribution for multikey hash files and node partitioning schemes for B-trees to enhance access concurrency. Some areas of future research that extend the work in this thesis are described below.

(1) We have proposed a general model and two stage processing model. Though these models are independent of the parallel processing architecture, they need to be tuned and further subdivided into several prototypes to reflect hardware architectures such as cube type interconnection.

(2) In many cases we do not know what the best achievable distribution is for a given file systems. It is worth while to investigate the existence of optimal distribution for these file systems. Knowledge of the existence of optimal distribution will help to develop methods for optimal distribution for these file systems.

(3) FX distribution methods have been developed for file systems in which each field size and the number of parallel access nodes are power of 2. FX distribution methods need to be extended for non power of 2 file systems.

(4) PNB-trees are developed for files on secondary storage devices. The application of PNB-tree approach to main memory databases needs to be investigated. Performance analyses of PNB-trees are mainly based on disk access time because disk access time is the most important parameter for the performance of external databases. However, performance analyses of PNB-trees in main memory databases will require different set of parameters.

LIST OF REFERENCES

LIST OF REFERENCES

[Aho79] Aho, A.V. and Ullman, J.D., "Optimal Partial-Match Retrieval When Fields Are Independently Specified," *ACM Trans. on Database Systems*, Vol. 4 No. 2 , June 1979, pp. 168-179.

[Amm85] Ammann, A., Hanrahan, M. and Krishnamurthy, R.,"Design of a Memory Resident DBMS,' *Proc. IEEE COMPCON*, 1985, pp. 54-57.

[Ban79] Banerjee, J., Hsiao, D.K. and Kannam, K., "DBC - A Database Computer for Very Large Databases," *IEEE Trans. on Computers*, Vol. c-28, No. 6, June 1979, pp. 414-429.

[Bat77] Batcher, K.E., "The Multidimensional Access Memory in STARAN", *IEEE Trans. on Computers*, Feb. 1977, pp. 174-177.

[Bay72] Bayer, R. and McCreight, E.M., "Organization and Maintenance of Large Ordered Indices," *Acta Informatica*, 1:3, 1972, pp. 173-189.

[Bay79] Bayer, R. and Schkolnick, H., "Concurrency of Operations on B-trees," *Acta Informatica*, Vol. 9, 1977, pp. 1-21.

[Ben75] Bentley, J.L., "Multi-dimensional Binary Search Trees Used for Associative Searching," *Commu. of the ACM* Vol. 18, No. 9, 1975, pp. 509-517.

[Ben79] Bentley, J.L. and Friedman, J.H., "Data Structures for Range Searching," *ACM Computing Surveys*, Vol. 11, No. 4, Dec. 1979, pp. 397-409.

[Bic83] Bic, L. and Hartman, R.L., "A Network Oriented Dataflow System," *Database machines*, Leilich, H.O. and Missikoff, M., eds., Springer-Verlag, 1983, pp. 166-187.

[Bol79] Bolour, A., "Optimality Properties of Multiple-key Hashing Functions," *JACM, Vol. 26, No. 2, April 1979, pp. 196-210.*

[Bol81] Bolour, A., "Optimal Retrieval Algorithms for Small Region Queries," *SIAM J. Comput.* Vol. 10, No. 4, Nov. 1981, pp. 721-741.

[Bor82] Boral, H. and Dewitt, D.J., "Applying Dataflow Techniques to Database Machines," *IEEE Computer,* August 1982, pp. 57-63.

[Bor83] Boral, H. and Dewitt, D.J., "Database Machines : An Idea Whose Time Has Passed? A Critique of the Future of Database Machines," *Database machines*, Leilich, H.O. and Missikoff, M., eds., Springer-Verlag, 1983, pp. 166-187.

[Bur76] Burkhard, W.A., "Hashing and Trie Algorithms for Partial Match Retrieval," *ACM Trans. on Database Systems*, Vol. 4, No. 2, June 1976, pp. 175-187.

[Cev88] Ceverance, C. and Pramanik, S., "A High Speed KDL-RAM File System For Parallel Computers," Techinical Report, Computer Science Department, Michigan State University, Sept. 10, 1988.

[Com79] Comer, D., "The Ubiquitous B-trees," *ACM Computing Surveys,* vol. 11, no. 2, June 1979, pp. 121-136.

[Dew79] Dewitt, D.J., "DIRECT-A Multiprocessor Organization for Supporting a Relational Database Management System," *IEEE Trans. on Computers,* June 1979, pp. 395-406.

[Dew86] Dewitt, D.J., Gerber, R.H., Graefe, G., Heytens, M., Kumar, K.B. and Muralikrishna, M., "GAMMA : A Performance Dataflow Database Machines," *Proc. Conf. on Very Large Data Bases,* 1986, pp. 228-237.

[Dew84] Dewitt, D.J., Katz, R., Olken, F., Shapiro, L, Stonebraker, M. and Wood, D., "Implementation Techniques for Main Memory Database Systems," *Proc. ACM SIGMOD,* 1984, pp. 1-8.

[Du85] Du, H.C., "On the File Design Problem for Partial Match Retrieval," *IEEE Trans. on Software Eng.* Vol. SE-11, No. 2, Feb. 1985, pp. 213-222.

[Du86] Du, H.C., "A Heuristic Disk Allocation Method for Binary Cartesian Product Files," *BIT* 1986, pp.138-147.

[DuS82] Du, H.C. and Sobolewski, J.S., "Disk Allocation for Cartesian Product Files on Multiple-Disk Systems," *ACM Trans. on Database Systems*, Vol. 7 No. 1, March 1982, pp.82-101.

[Ell80-a] Ellis, C., "Concurrent Search and Insertion in AVL Trees," *IEEE Trans. on Computers,* Vol. c-29, Sept. pp. 811-817.

[Ell80-b] Ellis, C., "Concurrent Search and Insertion in 2-3 Trees," *Acta Informatica,* Vol. 14, 1980, pp. 63-86.

[Fag79] Fagin, R., Nievergelt, J., Pippenger, N. and Strong, H.R., "Extendible Hashing - A Fast Access Method For Dynamic Files,"*ACM Trans. on Database Systems*, Vol. 4 No. 3, Sept. 1979, pp.315-344.

[Fan86] Fang, M.T., Lee, R.C.T. and Chang, C.C., "The idea of De-clustering and Its Applications," *Proc. Conf. on Very Large Data Bases,"* Aug. 1986, pp.181-188.

[Fre81] Fredman, M.L., "A Lower bound on the Complexity of Orthogonal Range Queries," *JACM*, Vol. 28, No. 4, Oct. 1981, pp. 696-705.

[Gar86] Garg, A.K. and Gotlieb, C.C., "Order-Preserving Key Transformations," *ACM Trans. on Database Systems*, Vol. 11, No. 2, June 1986, pp. 213 -234.

[GarL84] Garcia-Molina, H., Lipton, R.J. and Valdes, J., "A Massive Memory Machine," *IEEE Trans. on Computers,* Vol. c-23, No. 5, May 1984, pp. 391-399.

[Gli83] Glinz, M., "A Dataflow Retrieval Unit for a Relational Database Machine," *Database machines*, Leilich, H.O. and Missikoff, M., eds., Springer-Verlag, 1983, pp. 166-187.

[Haw82] Hawthorn, P.B. and Dewitt, D.J., "Performance Analysis of Alternative Database Machine Architecture," *IEEE Trans. on Software Eng.,* Vol. SE-8, Jan. 1982, pp. 61-75.

[Hil86] Hillyer, B., Shaw, D.E. and Nigam, A., "NON-VON's Performance on Certain Database Benchmarks," *IEEE Trans. on Software Eng.* Vol. SE-12, No. 4, April 1986, pp. 577-583.

[Hsi78] Hsiao, D.K. and Baum, R.I., "Concepts and Capabilities of a Database Computer," *ACM Trans. on Database Systems,* 1978, pp. 347-384.

[Hsi88] Hsiao, Y.-S. and Tharp, A.L., "Adaptive hashing," *Inform. Systems*, Vol. 13, No. 1, 1988, pp. 111-127.

[Kak85] Kakuta,T., Miyazaki,N., Shibayama,S., Yokota,H. and Murakami,K.,"The Design and Implementation of Relational Database Machine Delta," *Database Machines, Fourth International Workshop,* 1985, pp. 13-34.

[Kho88] Khoshafian, S., Valduriez, P. and Copeland, G., "Parallel Query Processing for Complex Objects," *Int'l Conf. on Data Engineering,* 1988, pp. 202 - 209.

[Kim86] Kim, M. Y., "Synchronized Disk Interleaving", *IEEE Trans. on Computers* Vol.c-35, No.11, Nov. 1986, pp. 978-988.

[Kim88] Kim, M. and Pramanik, S., "Optimal File Distribution For Partial Match Retrieval," *Proc. ACM SIGMOD,* 1988, pp. 173-182.

[Knu73] Knuth, D.E. The Art of Computer Programming, Vol. 3 : Sorting and Searching, Addison-Wesley, 1973, pp. 517-537.

[Kun80] Kung, H.T. and Lehman, P.L., "Concurrent Manipulation of Binary Search Trees," *ACM Trans. on Database Systems,* Vol. 5, No. 3, Sept. 1980, pp. 354-382.

[Lar78] Larson, P., "Dynamic Hashing," *BIT*, 1978, pp. 184-201.

[Law82] Lawrie, D.H. and Vora, C.R., "The Prime Memory System for Array Access," *IEEE Trans. on Computers*, May 1982, pp. 435-442.

[Lee77] Lee, D.T. and Wong, C.K., "Worst-Case Analysis for Region and Partial Region Searches in Multidimensional Binary Search Trees and Balanced Quad Trees," *Acta Informatica* Vol. 9, 1977, pp. 23-29.

[Leh81] Lehman, P.L. and Yao, S.B., "Efficient Locking for Concurrent Operations on B-Trees," *ACM Trans. on Database Systems,* Vol. 6, No. 4, Dec. 1981, pp. 650-670.

[Leh86] Lehman, T.J. and Carey, M.J., "Query Processing in Main Memory Database Management Systems," *Proc. ACM SIGMOD,* 1986, pp. 239-250.

[Lei78] Leilich, H.O., Stiege, G. and Zeidler, H.Ch., "A Search Processor for Database Management Systems," *Proc. of the 4th Int'l Conf. on Very Large Data Bases,* 1978, pp. 280-287.

[Lel85] Leland, M.D. and Roome, W.D., "The Silicon Database Machine" *Database Machines, Fourth International Workshop*, 1985, pp. 169-189.

[Lit80] Litwin, W., "Linear Hashing : A New Tool for File and Table Addressing," *Proc. of the 6th Int'l Conf. on Very Large Data Bases*, 1980, pp.212-223.

[Mis83] Missikoff, M.J. and Terranova, M., "The Architecture of Relational Database Computer Known as DBMAC," *Advanced Database Machine*

Architecture, D.k. Hsiao(ed), Prentice Hall, 1983, pp. 109-129.

[Ozk75] Ozkarahan, C.A., Schuster,S. A., and Smith, K. C., "RAP-An Associative Processor for Database Management," *Proc. NCC, AFIPS*, 1975, pp. 379-387.

[Pra86] Pramanik, S. and Davis, H., "Multi Directory Hashing," Technical Report, Computer Science Department, Michigan State University, Oct. 15, 1986.

[Pra87] Pramanik,S. and Kim, M., "HCB_tree : A B_tree Structure for Parallel Processing," *Proc. Int'l Conf. on Parallel Processing,* 1987, pp. 140-146.

[Pra88-a] Pramanik, S. and Kim, M., "HCB_tree : A Height Compressed B_tree for Parallel Processing," *Information Processing Letters*, November 1988, pp. 213-220.

[Pra88-b] Pramanik, S. and Kim, M.,"Generalized Parallel Processing Models for Database Systems," *Proc. Int'l Conf. on Parallel Processing,* 1988, pp. 76-83.

[Pra88-c] Pramanik, S. and Kim, M., "Parallel Processing of Large Node B-trees," *IEEE Trans. on Computers* (to be published).

[Pra88-d] Pramanik, S. and Kim, M., "Optimal Data Distribution for Parallel Processing of Multiattribute Range Queries," Technical Report, Computer Science Department, Michigan State University, Nov. 15, 1988.

[Riv76] Rivest, R.L., "Partial-Match Retrieval Algorithms," *SIAM J. Computing*, Vol.5, No.1, March 1976, pp. 19-50.

[Ros87] Rosenau, T. and Jajodia, S., "Parallel Relational Database Operations on the Butterfly Parallel Processor : Projection Results," Technical Report, Naval Research Laboratory, July 1987.

[Ross85] Ross, S.M., *Introduction to probability Models,* Academic Press, 1985, 3rd Edition, pp. 355-399.

[Rot74] Rothnie, J.B.Jr. and Lozano,T.,"Attribute based file organization in a paged memory environment," *Comm. ACM,* Vol.17, No.2, 1974, pp. 63-69.

[Sam76] Samadi, B., "B-trees in A System with Multiple Users," *Information Processing Letters*, October 1976, pp. 107-112.

[Sch83] Schweppe,H., Zeidler,H.Ch., Hell,W., Leilich,H.O., Stiege,G. and Teich,W.,"RDBM-A Dedicated Multiprocessor System for Database Management," *Advanced Database Machine Architecture,* Hsiao, D.K. ed., Prentice Hall, 1983, pp. 36-86.

[Slo70] Slotnick, D.L., "Logic Per Track Devices," *Advances in Computers,* Vol. 10, Academic Press, 1970, pp. 291-296.

[Sto87] Stone, H., "Parallel Querying of Large Databases : A Case Study," *IEEE Computer,* Oct. 1987, pp. 11-21.

[Su79] Su, S.Y.W., Nguyen, L.H., Eman, A. and Lipovski, G.J. "The Architectural Features and Implementation Techniques of multicell CASSM," *IEEE Trans. on Computers,* June 1979, pp. 430-445.

[Sun85] Sung, Y.Y., "Parallel Searching for Binary Cartesian Product Files," *Proc. ACM CSC Conf.,* 1985, pp. 163-172.

[Sun87] Sung, Y.Y., "Performance Analysis of Disk Modulo Allocation Method for Cartesian Product Files," *IEEE Trans. on Software Eng.*, Vol. SE-13, No. 9, Sept. 1987, pp. 1018-1026 .

[Yao78] Yao, A.,"Random 2-3 trees," *Acta Informatica,* Vol. 9, 1978, pp. 159-170.