A SYSTEMATIC APPROACH TO THE MANAGEMENT OF DATA ON DISTRIBUTED DATA BASES

Benjamin Wan-Sang Wah

Ph.D.

Electrical Engineering
and Computer Science

Sponsors: Ballistic Missile Defense
Army Research Office
National Science Foundation

C. V. Ramamoorthy
Chairman of Committee

ABSTRACT

Recent studies have revealed that the design of a distributed data base management system is a major source of difficulty in designing a distributed computer systems. Research has involved an investigation of the data management issues, examining in particular the *query decomposition*, the *file placement*, the *task scheduling*, and the *hardware support* issues. The inherent relationships among these issues are analyzed and a unified approach is provided to design data management strategies on distributed data bases.

One of the major problems in distributed computer systems is the minimization of communication overheads among nodes. This is the objective of the study in query decomposition and file placement and migration. Two complementary techniques are developed in query decomposition so that non-decomposable queries which require the use of multiple files can be decomposed into multiple sub-queries which require the use of single file. The communication overheads are reduced because the queries do not have to be processed at a common location and can be distributed to the different nodes on the distributed system. The study of query decomposition has also shown that the placements of multiple files can be decomposed into the multiple sub-problems, one for each file. An investigation is made on the file placement problem, with the

objective of minimizing the overall storage, migration, updating and operational costs on the system. By showing that the file placement problem and the facility location problem are isomorphic, many results derived in one problem can be applied to solve the other problem. Further, some results derived in one problem can be shown to be weaker than the corresponding results derived in the other problem. The last two areas of study are related to the distributed scheduling of tasks on distributed systems and the design of the necessary hardware support for data management. The task scheduling problem for a distributed system is shown to be NP-complete. However, an optimal average algorithm is developed for a restricted class which minimizes the expected completion time for a set of random requests. In the hardware support issue, the design of an associative memory which is capable of equality, proximity, threshold and extremum searches is investigated. The complexity of the design is 17 gates per cell. Its extensions to the design of associative sequential memories and data base machines are developed.

ACKNOWLEDGEMENT

The author would like to express his sincere gratitude to a number of individuals for their help in the research and preparation of this dissertation. Special thanks is due to Professor C. V. Ramamoorthy who was his thesis supervisor and the dissertation committee chairman. Throughout the years, Professor Ramamoorthy has been a constant source of advice, guidance, support and encouragement. He was and will be an ideal for the author to follow. Without his support, none of these would have been possible.

The author would also like to thank Professors D. Ferrari and I. Adler for reading the thesis, providing many helpful and valuable advice and encouragement, and serving on the thesis committee. Special thanks are also due to Miss Y. W. Ma, for reading and typing part of the draft and criticizing on the research. She has also been a source of spiritual support and encouragement througout these years. In addition, his colleagues, Messers F. Bastani, F. Ho, G. Ho, J. Favaro, C. Jen, T. Krishnarao, F. Leung, R. Mok, C. Nam, H. H. So and K. Wu are thanked for providing many helpful comments and a friendly environment for research.

He also wants to thank the Ballistic Missile Defense, the Army Research Office and the National Science Foundation for its support through contracts DASG-60-77-C-0138, DAAG29-78-G-0189 and grants MCS-77-27293, MCS-77-28361. Messers C. R. Vick and J. E. Scalf are also thanked for their many helpful discussions.

Finally and most importantly, the author wishes to thank his parents in Hong Kong for their unselfish and devoted support, both spiritually and finally.

This thesis was typed on a DEC VAX 11/780 computer supported by NSF grant MCS78-07291.

TABLE OF CONTENTS

PAGE

Chapter 1 INTRODUCTION 1

1.1 What is a Distributed Data Base 1

1.2 Issues in Designing Data Base Systems 3

1.2.1 Issues in Logical Organization.. 4

1.2.2 Issues in Architecture... 6

1.2.3 Issues in Operational Control.. 7

1.2.4 Issues in Evolution 9

1.3 Architecture of the System supporting a DDB 16

1.4 Objectives and Contributions of the Research 21

1.4.1 Problem Statement 22

1.4.2 Approach.. 22

1.4.3 Contributions of this Research 26

Chapter 2 QUERY DECOMPOSITION ON A

DISTRIBUTED RELATIONAL DATA BASE 28

2.1 Queries on a Relational DB 28

2.2 The Placements of Relations on a DDB 32

2.3 Cost Reduction on the Placements of Relations on a DDB

by utilizing Redundant Information 40

2.4 A Numerical Example to illustrate the Use of Redundant

Information on a DDB.. 49

2.5 Cost Reduction on the Placements of Relations on a DDB

by File Partitioning 53

2.6 A Numerical Example to Illustrate the Use of File

Partitioning on a DDB.. 58

2.7 Cost Reduction on the Placement of Relations on a DDB by utilizing Redundant Information and File Partitioning 59

2.8 A Numerical Example to illustrate the use of Additional Redundant Information and File Partitioning on a DDB 63

2.9 Conclusion 69

Chapter 3 THE PLACEMENT AND MIGRATION OF MULTIPLE COPIES OF A FILE ON A DCS 71

3.1 Introduction 71

3.2 Definition of the Problem 71

3.3 Motivations for File Placement and Migration 73

3.4 Previous Work on the File Allocation Problem 74

3.5 Previous Work on the Single Commodity Warehouse Location Problem 80

3.6 The Isomorphism between File Allocation and Single Commodity Warehouse Location 84

3.7 Implications of the Isomorphism between the FAP and the SCWLP 84

3.8 A Heuristic for the FAP - Algorithm 3.1 91

3.9 DFAP - The Migration of Files on a DCS 103

3.10 Conditions to reduce the Complexity of DFAP 106

3.10.1 The Problem of selecting the Times for Migration is NP-Complete 106

3.10.2 Criteria for initiating a Migration 108

3.11 Conclusion 114

Chapter 4 TASK SCHEDULING ON DISTRIBUTED COMPUTER SYSTEMS 116

4.1 Introduction 116

4.2 A Model for the scheduling of Tasks on Distributed Systems 119

4.2.1 The Model 119

4.2.2 Assumptions which allow the Task Scheduling Problem to be simplified 125

4.3 NP-completeness of the Task Scheduling Problem 128

4.4 The Restricted Model - An Optimal Algorithm for scheduling Requests on an Interleaved Memory System 137

4.4.1 Requirements for the Design of a Primary Memory 137

4.4.2 Characteristics of the Access Sequence of a Pipelined Processor 139

4.4.3 Previous work on the Study of Interleaved Memories 141

4.4.4 The Organizations of Primary Memory for a Pipelined Processor 145

4.4.5 Optimality of the MWFMF Scheduling Algorithm 157

4.4.6 Embedded Markov Chain Technique 172

4.4.7 Simulation Technique 178

4.4.7.1 Simulation Results 178

4.4.7.2 Application of Multiple Linear Regression to obtain a Closed Form Formula 182

4.4.8 Effects of separating the Instruction and the Data Area 193

4.4.9 Degradation in Performance due to Dependencies 196

4.4.9.1 The Model used to estimate the Performance due to Dependencies 196

4.4.9.2 Computation of Degradation in Performance 199

4.4.10 Some Final Remarks about the Design of Interleaved Memories 208

4.5 A Heuristic for the scheduling of Tasks on the General Model 212

4.6 Conclusion 216

Chapter 5 HARDWARE SUPPORT FOR DATA MANAGEMENT ON DISTRIBUTED COMPUTER SYSTEMS 218

5.1 Introduction 218

5.2 A Design of a Fast Cellular Associative Memory for Ordered Retrieval 220

5.2.1 Previous Work 220

5.2.2 Symbols used in the Design 222

5.2.3 Basic Associative Memory Organization 224

5.2.4 Definition of Search Operations 226

5.2.5 Algorithms and Implementations of Basic Searches 228

5.2.5.1 Mode A: Equality-Proximity-Threshold Search Mode. 229

5.2.5.2 Mode B: Least Value Search Mode 233

5.2.5.3 Mode C: Greatest Value Search Mode 240

5.2.6 Ordered Retrieval 242

5.2.7 Some Speed-up Techniques 244

5.2.8 Issues and Limitations 252

5.2.8.1 LSI Implementation 252

5.2.8.2 Manufacturing Defects 253

5.2.8.3 Modular Expansion 254

5.2.8.4 Multiple Match Resolution 258

5.2.9 Comparisons with other Methods of Ordered Retrieval 259

5.3 Data Base Machines 263

5.3.1 Introduction 263

5.3.2 Issues in the Design of Data Base Machines 266

5.3.3 Classification of Data Base Machines 268

5.3.4 Extension of the Associative Memory Design to Sequential Memories 276

5.4 Conclusion 280

Chapter 6 CONCLUSION 282

Appendix A THE ISOMORPHISM BETWEEN PROCESS ALLOCATION PROBLEM AND THE SINGLE COMMODITY QUADRATIC ASSIGNMENT PROBLEM 285

Appendix B THE LINEAR PROGRAMMING LOWER BOUND OF A CANDIDATE PROBLEM 287

Appendix C THE EXPECTED VALUE OF A CANDIDATE PROBLEM....................... 289

Appendix D BIBLIOGRAPHY 291

LIST OF FIGURES

PAGE

Figure 1.1 Classification of Issues in Distributed Data Base Systems............ 5

Figure 1.2 Architecture of a DDB System.. 10

Figure 1.3 The Density Growth of Large Scale Integrated Circuits................ 11

Figure 1.4 The Exponential Growth of CPU Speed 13

Figure 1.5 Availability of Memory Technologies.. 17

Figure 1.6 Storage Hierarchy.. 19

Figure 1.7 Architecture of a Data Base Machine 20

Figure 1.8 Relationships among Various Data Management Issues 25

Figure 2.1 Relations S and SP (not all domains are shown).......................... 29

Figure 2.2 Retrieval and Update Rate on a 2-Relation DDB from Node i......... 34

Figure 2.3 Relations S and SP with (S.s#=SP.s#) information compiled into the Relations... 41

Figure 2.4 Retrieval and Update Rates on a 2-relations DDB from Node i using Additional Redundant Information.................................. 45

Figure 2.5 A Plot of Cost Ratio w.r.t. γ for Various Values of δ under the use of Redundancy.. 51

Figure 2.6 A Plot of Cost Ratio w.r.t. η for Various Values of δ under the use of Redundancy.. 52

Figure 2.7 The Retrievals and Updates on a DDB (2 nodes) with and without File Partitioning... 54

Figure 2.8 A Plot of Cost Ratio w.r.t. fu^1 for Various Values of fu^2 under File Partitioning... 60

Figure 2.9 A Plot of Cost Ratio w.r.t. γ for Various Values of δ under the use of Redundancy and Partitioning (fu^1=fu^2=0.75) 65

Figure 2.10 A Plot of Cost Ratio w.r.t. η for Various Values of δ under the use of Redundancy and Partitioning ($fu^1=fu^2=0.75$) 66

Figure 2.11 A Plot of Cost Ratio w.r.t. fu^1 for Various Values of fu^2 under the use of Redundancy and Partitioning ($\delta=\eta=\gamma=0.5$) 67

Figure 2.12 A Plot of Cost Ratio w.r.t. fu^1 for various values of fu^2 under the use of Redundancy and Partitioning ($\delta=\eta=\gamma=0.4$) 68

Figure 3.1 File Assignment Algorithm - Algorithm 3.1 93

Figure 3.2a Evaluation of Casey's 5 node Example using MINLB 96

Figure 3.2b Evaluation of Casey's 5 node Example using MINE 97

Figure 3.2c Evaluation of Casey's 5 node Example using MAXDLB 98

Figure 3.2d Evaluation of Casey's 5 node Example using MAXDE 99

Figure 4.1 Task Precedence Graph for the Processing of a Query which requires the Use of Geographically Distributed Files.......... 118

Figure 4.2 Model of an SIMD Computer System 120

Figure 4.3 Conceptual Model of a DCS 121

Figure 4.4 An SIMD Model for Task Scheduling on a DCS 123

Figure 4.5 Precedence Graph of Tasks for Request i which can be scheduled on the SIMD Model 124

Figure 4.6 Proof of Theorem 4.1 130

Figure 4.7 Timing Diagram for the Proof of Theorem 4.2 132

Figure 4.8 Timing Diagram for the Proof of Theorem 4.5 136

Figure 4.9 Organization I - A Model of Interleaved Memories with a Single Request Queue 147

Figure 4.10 A Gantt Chart to illustrate the Operation of the Interleaved Memories in Staggered Cycles (m=4) 148

Figure 4.11 MWFMF Scheduling Algorithm - Algorithm 4.3 151,2

Figure 4.12 Organization II - A Model of Interleaved Memories with

Multiple Request Queues 154

Figure 4.13 The Improvement of Average Memory Utilization with Buffer Size for Org. I (m=8) 188

Figure 4.14 The Increase of Average Waiting Cycles with Buffer Size for Org. I (m=8) 189

Figure 4.15 The Decrease of Average Memory Utilization w.r.t. Degrees of Interleaving for Org. I with MWFMF Scheduling Algorithm 190

Figure 4.16 The Decrease of Waiting Cycles w.r.t. Degrees of Interleaving for Org. I with MWFMF Scheduling Algorithm 191

Figure 4.17 The Average Buffer Utilization for Org. II (m=8) 192

Figure 4.18 The Average Buffer Utilization vs. the Degrees of Interleaving for Org. II with MWFMF Scheduling Algorithm (m=8) 194

Figure 4.19 Model of CPU-Memory used for Estimation of the Effects of Dependency 198

Figure 4.20 Histogram showing the Statistics of Dependent Events in Traces used 200

Figure 4.21 Graph showing the Cumulative Probability of Number of Instructions executed between 2 Conditional Jumps 201

Figure 4.22 The Variation of the Number of Requests in the Buffers 202

Figure 4.23 Buffer Utilization under Dependency for Org. I with a MWFMF Algorithm (Trace Driven Simulations) 209

Figure 4.24 Simulation Results for Algorithm 4.4 using 1000 Samples of 7 Randomly Generated Jobs (m=2, p=2) 215

Figure 5.1 Cellular Logic Associative Memory Block Diagram 225

Figure 5.2 Bit-cell with Equality, Greater-than, Less-than and Proximity Capability for Mode A Operation 232

Figure 5.3 Bit-cell with Least Value Search Logic for Mode B Operation 235

Figure 5.4 Bit-cell for Simultaneous Ascending Order Retrieval and Descending Order Retrieval or Equality-Threshold-Proximity Searches 241

Figure 5.5 Flow Chart for Ascending Order Retrieval 243

Figure 5.6 Bit-cell j and j+1 of word i with Equality-Proximity Search and Lookahead Logic 246

Figure 5.7 Modular Extension for Proposed Associative Memory 256

Figure 5.8 Flow Chart for Ascending Order Retrieval of m^2 words in a 3-dimensional Associative Memory [see Fig. 5.7c] 257

Figure 5.9 Comparison of Retrieval Speeds for a 5 bit Tag with k Words Flagged 264

Figure 5.10 Backend Systems using Conventional Mini-computers 269

Figure 5.11 Intelligent Controllers - Cellular Logic 271

Figure 5.12 Intelligent Controllers - Associative Memories 274

Figure 5.13 Intelligent Controllers - MIMD Architecture 275

Figure 5.14 Associative Sequential Memory 278

Figure 5.15 Associative Logic for Associative Sequential Memory 279

LIST OF TABLES

PAGE

Table 1.1 Typical Ballistic Missile Defense Data Base Processing Requirements in a Centralized Environment.................................. 2

Table 1.2 Typical Values for LSI Semiconductor RAM's (1978) 12

Table 3.1 Examples of Communication Costs on Telenet Data Communication Network (July 1, 1978)... 74

Table 3.2 A Summary of the Previous Work in File Placement/Migration...... 76

Table 3.3 Mapping between the Defined Notations in this Thesis and Casey's Notations... 77

Table 3.4 Mapping between the (D)FAP and the SC(D)WLP 85

Table 3.5 Summary of Conditions for Placement and Non-Placement of a file at Node $i \in K_2$.. 89

Table 3.6 % Deviations of File Allocation Heuristic from Optimal Solutions ... 101

Table 3.7 Execution Time of Heuristic in Seconds on the CDC 6400.............. 102

Table 4.1a Simulation Results for Org. I with RR Algorithm............................ 179

Table 4.1b Simulation Results for Org. I with FFF Algorithm.......................... 180

Table 4.1c Simulation Results for Org. I with MWFMF Algorithm 181

Table 4.2a Simulation Results for Org. II with RR Algorithm 183

Table 4.2b Simulation Results for Org. II with FFF Algorithm.......................... 184

Table 4.2c Simulation Results for Org. II with MWFMF Algorithm 185

Table 4.3 Coefficients of 3rd Order Polynomial Regression of Organizations I and II under MWFMF Scheduling Algorithm 186

Table 4.4 Comparison between Merged and Separated Instruction-Data Areas for Org. I - Trace Driven Simulations.................................... 195

Table 4.5 Comparison between Merged and Separated Instruction-Data

Areas for Org. II - Trace Driven Simulations 195

Table 5.1 A Comparison Table for Ordered Retrieval Schemes 262

Table A-1 Mapping between Stone's Process Allocation Problem and the Single Commodity Quadratic Assignment Problem 286

1. *INTRODUCTION*

The recent advances in large scale integrated logic and communication technology, coupled with the explosion in size and complexity of the application areas, have led to the design of distributed architectures. Basically, a *Distributed Computer System (DCS)* is considered as an interconnection of digital systems called *Processing Elements (PEs)*, each having certain processing capabilities and communicating with each other. This definition encompasses a wide range of configurations from an uniprocessor system with different functional units to a multiplicity of general purpose computers (e.g. ARPANET). In general, the notion of "distributed systems" varies in character and scope with different people [RAM76]. So far, there is no accepted definition and basis for classifying these systems. In this thesis, we limit our discussion to a class of DCS's with an interconnection of dedicated/shared, programmable, functional PEs and working on a set of jobs which may be related or unrelated.

1.1 *WHAT IS A DISTRIBUTED DATA BASE*

Due to the information explosion and the need for more stringent requirements, the design of efficient coordination schemes for the management of data on a DCS is a very critical problem. To indicate the amount of data processed, the typical data base processing requirements for a ballistic missile defense system [DDP78], operating in a centralized environment are shown in Table 1.1. In order to manage the data on a computer system (centralized or distributed) and satisfy all the requirements, systematic techniques must be developed so that the system can be realized in a cost-effective way.

Data on a DCS are managed through a *Data Base (DB)*, which is a collection of stored operational data used by the application systems of some particular enterprise [DAT77, FRY76]. A *Distributed Data Base (DDB)* can be thought of as objective of minimizing the overall storage, migration, updating and operational costs on the system. By showing that the file placement problem and the facility location problem are isomorphic, many results derived in one problem can be applied to solve the other problem. Further, some results derived in one problem can be shown to be weaker than the corresponding results derived in the other problem. The last two areas of study are related to the distributed scheduling of tasks on distributed systems and the design of the necessary hardware support for data management. The task scheduling problem for a distributed system is shown to be NP-complete. However, an optimal average algorithm is developed for a restricted class which minimizes the expected completion time for a set of random requests. In the hardware support issue, the design of an associative memory which is capable of equality, proximity, threshold and extremum searches is investigated. The complexity of the design is 17 gates per cell. Its extensions to the design of associative sequential memories and data base machines are developed.

programs, together with the sub-schema, collectively form the Data Base Management System [FRY76, BAC75]. The Data Base Management System allows data sharing among a community of users, while insuring the integrity of the data over time, and providing security against unauthorized access. It also provides the transparency of the data, in order to allow the data to be stored in different formats in different parts of the system. Finally, it provides an interface between the users and the system.

The data base can be classified according to how these components are put together. In [ASC74], two classifications are proposed, the first is based on the number of Data Base Management Systems in the network and the second is based on the centralization or decentralization of the file directory and the data. In [BOO76], the DDB's are classified into two structures, partitioned data bases and replicated data bases. A partitioned data base is one that has been decomposed into physically separate units, and distributed across multiple nodes of a DCS. The partitioning will normally be based on the distribution of access requirements. In a replicated data base, all or part of the data base is replicated at multiple processing nodes. The amount of partitioning and replication depends on the architecture of the distributed system, the amount of traffic anticipated and other requirements such as reliability, security, etc.

1.2 *ISSUES IN DESIGNING DISTRIBUTED DATA BASE SYSTEMS*

The issues associated with the design of a DDB can be classified from an user's viewpoint or from a system designer's viewpoint. From an user's viewpoint, the users are concerned with the type of organization and controls which can give efficient and reliable operations and can satisfy their requirements. The users usually do not relate very closely other factors such as technology and architecture in their considerations. On the other hand, from a designer's viewpoint, the designers are more concerned with the architecture of the system and its dependency on technology. However, the issues considered from both viewpoints are not independent and must be investigated jointly in the design of a DDB. We have therefore taken an integrated approach and have classified these issues into four categories. The classification is shown in Figure 1.1.

1.2.1 *Issues in Logical Organization*

These issues are related to the user-system interface and can be classified as:

(A) *User Interface*

The user interface may be defined as a boundary in the system below which everything is invisible to the user [DAT77]. The function of this interface is to provide the users with an efficient and powerful query language and to help the users to manipulate the data in the DB. The query language must be powerful enough so that an entire set can be manipulated as a single object, instead of being restricted to one record at a time. The complexity of this interface depends on the required ease with which users wish to access the data and it directly governs the design of communication processors.

(B) *Data Base Organization*

A data base is generally organized in one or more of the data models: relational, hierarchical or network model, where a data model refers to a representation of the entire information content of the DB in a form that is somewhat abstract in comparison with the way in which data is physically stored [DAT77]. There are other models like the binary association model and the external set model which are not quite popular. Each user views the data base through an external model which may be one of the above data models. The data base should therefore be able to support multiple data models for different users and

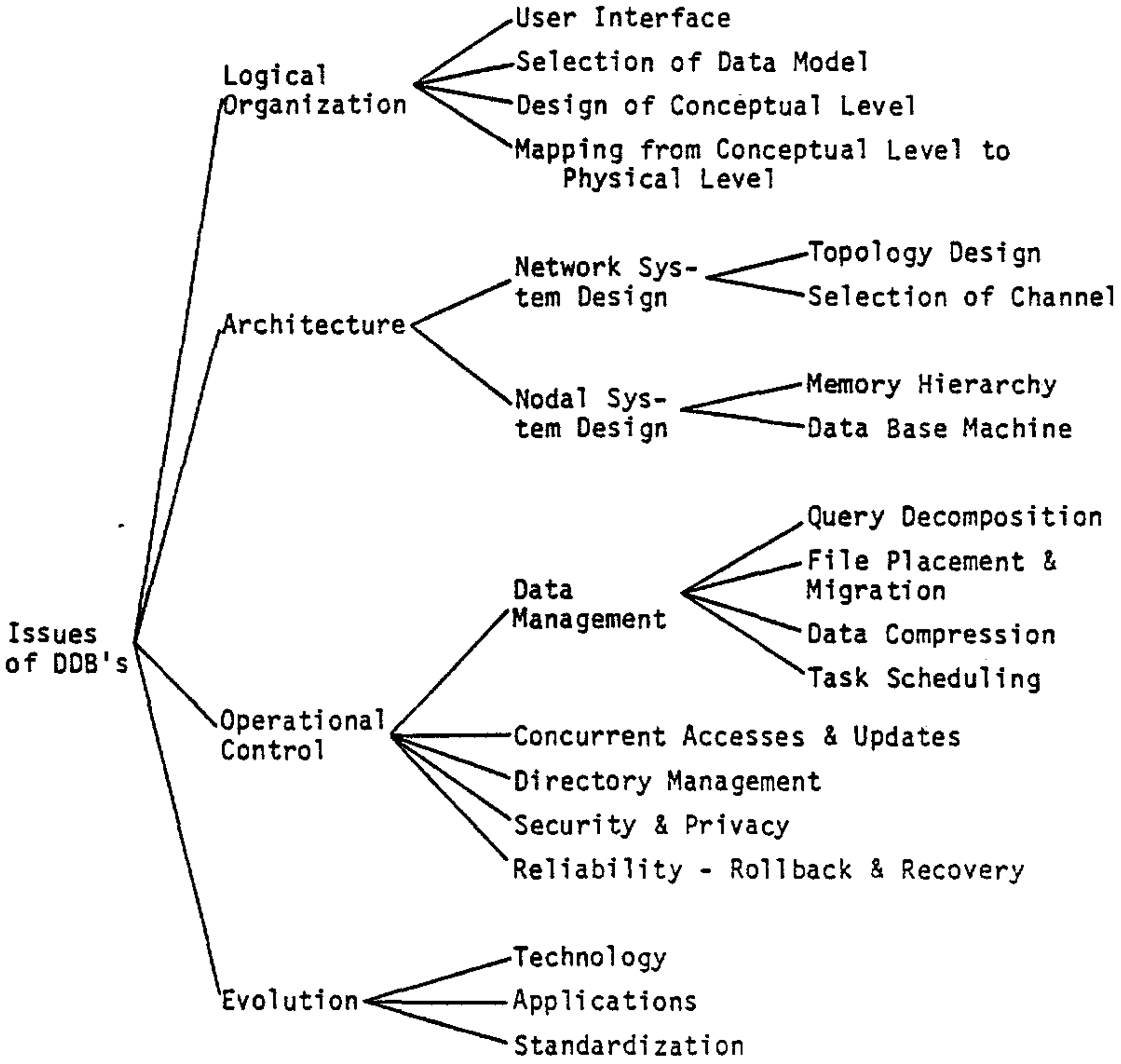


Figure 1.1 Classification of Issues in Distributed Data Bases

to provide users with transparent accesses. The efficiency of a DDB is very much dependent on the type of organization since it affects the storage organization, access mechanisms and the communication requirements. The criteria for designing and selecting a model has not yet been well understood or established, nor is it likely to be established in the near future. The designers of a DDB are therefore confronted with two decisions: which data model to utilize and how to structure the data for a chosen model [SIL76]. Further, there is the problem of mapping the different external models onto the conceptual level.

(C) *Design of the Conceptual Level*

The conceptual level is a level of indirection between the external level which consists of different data models and language interfaces and the internal level which consists of the physically stored data. The conceptual level actually maps the users' views onto physical data and is intended to provide a solid and enduring foundation for the total operation of the DDB. Its design depends on how the data are stored, the physical storage media, the number of different data models, the way that data are distributed on the DCS and other user requirements. It is important to construct a conceptual schema at a suitable level of abstraction in the design stage [DAT77]. Many of the techniques in artifical intelligence have been applied successfully in this design.

1.2.2 *Issues in Architecture*

(A) *Network System Design*

The DCS is made up of nodal processors interconnected together through an interconnection network. There are many data base related issues associated with the design of network systems in addition to the design issues of efficient nodal systems. Among these are: the selection of network topology to support DDB requests; the selection of the channel type; the design of network control strategies; the design of communication processors, etc. Some of these issues have been studied in [RAM76, RAM79b].

(B) *Nodal System Design*

The design of the nodal architecture to support a DDB is concerned with the design of fast storage sub-system whose function is to provide the nodal processor sub-system and users with fast retrievals and accesses to the stored data. The storage sub-system usually consists of a memory hierarchy that is divided into levels. These levels are made up of memory elements of varying speeds and the fastest level is interfaced to the processor sub-system. Further, intelligence have also been distributed to the various levels of the hierarchy. One such design is the data base machine [HSI77]. Issues like the selection of the number of levels and the size of each level of the memory hierarchy; the design of virtual memory for automatic file management; the utilization of new memory technologies; the hardware design for supporting data base operations in a data base machine; the interconnection structure between memories and processors; etc. must be considered in the design.

1.2.3 *Issues in Operational Control*

These issues are concerned with the efficient, correct, reliable and secure operations of the data base. They can be classified into:

(A) *Resource Management of Data*

These are issues related to the management of data and files as resources of the system so that multiple users can share the files on the data base efficiently [RAM79a]. The control of files as resources is not only applied at the file level, where the files have to be placed at nodes easily accessible to users and the data have to be compressed for efficient communication and storage, but it ranges from the users' level to the physical level. On the users' level, the queries have to be processed so that the amount of data movements is minimum. On the physical level, the individual file requests have to be sequenced so that maximum hardware parallelism can be achieved. Some of these issues are the focus of study in this thesis.

(B) *Concurrent Accesses and Updates*

In a DDB where users share the same data, there are several problems associated with multiple accesses and updates. When users try to access the common data, there would be interference among the accesses, and the communication protocol should be designed to minimize this interference. Another problem related to consistency arises when data elements with multiple copies at different locations are to be updated. Simple locking mechanisms cause excessive delays and may cause throughput degradation in the DCS. Efficient updating schemes are needed and the architectures would be very much influenced by such schemes [ESW76].

(C) *Directory Management*

The directory is a special file in which the addresses for various files on the system are provided. Each access to a file must therefore pass through the directory. Due to the high intensity of the accesses on the directory, special attention must be paid to its design. In particular, the designer has to consider the type of directory structure which is most suitable for his application and whether the directory should be replicated or partitioned. In general, a combination of replication and partition is used. Further, reliability considerations must be made in the design of the directory [ROT77].

(D) *Security and Privacy*

Another important issue in the design of a DDB is security and privacy. Security refers to the protection of data against deliberate or accidental destruction, unauthorized access or modification of data. On the other hand, privacy refers to the right of an individual user to determine for himself what personal information to share with others as well as what information to receive from others. As the size of the data base increases, the threat to security and privacy increases. In addition, it is increasingly difficult to implement effective measures in a DDB. Additional techniques such as data encryption would affect the transmission efficiency and the communication mechanisms [BAD78, DOW77].

(E) *Reliability - Rollback and Recovery*

The determination of the necessary hardware for reliable operations, the data redundancy and the reconfiguration strategies are another major issue in the design of a DDB. Multiple copies of data base realm offer fast recovery; checkpointing of realms, dumping and journal rollback and roll-forward offer a slower but cheaper recovery. The effect of any recovery mechanism and reconfiguration strategy on the response time and the associated overhead must be weighed against the reliability requirements [KRI78].

1.2.4 *Issues in Evolution*

In order for the system to be able to adapt to new application requirements and technology advancements, evolutionary measures must be incorporated into the system at the design stage. Three of the contradicting issues of evolution are:

(A) *Technology Dependence*

Technology is one of the most important driving force for the success of a computer system. As seen in Figures 1.2 and 1.3, the number of components per chip is approximately doubling each year, and the CPU speed is growing exponentially each year. These faster and denser logic, together with a variety

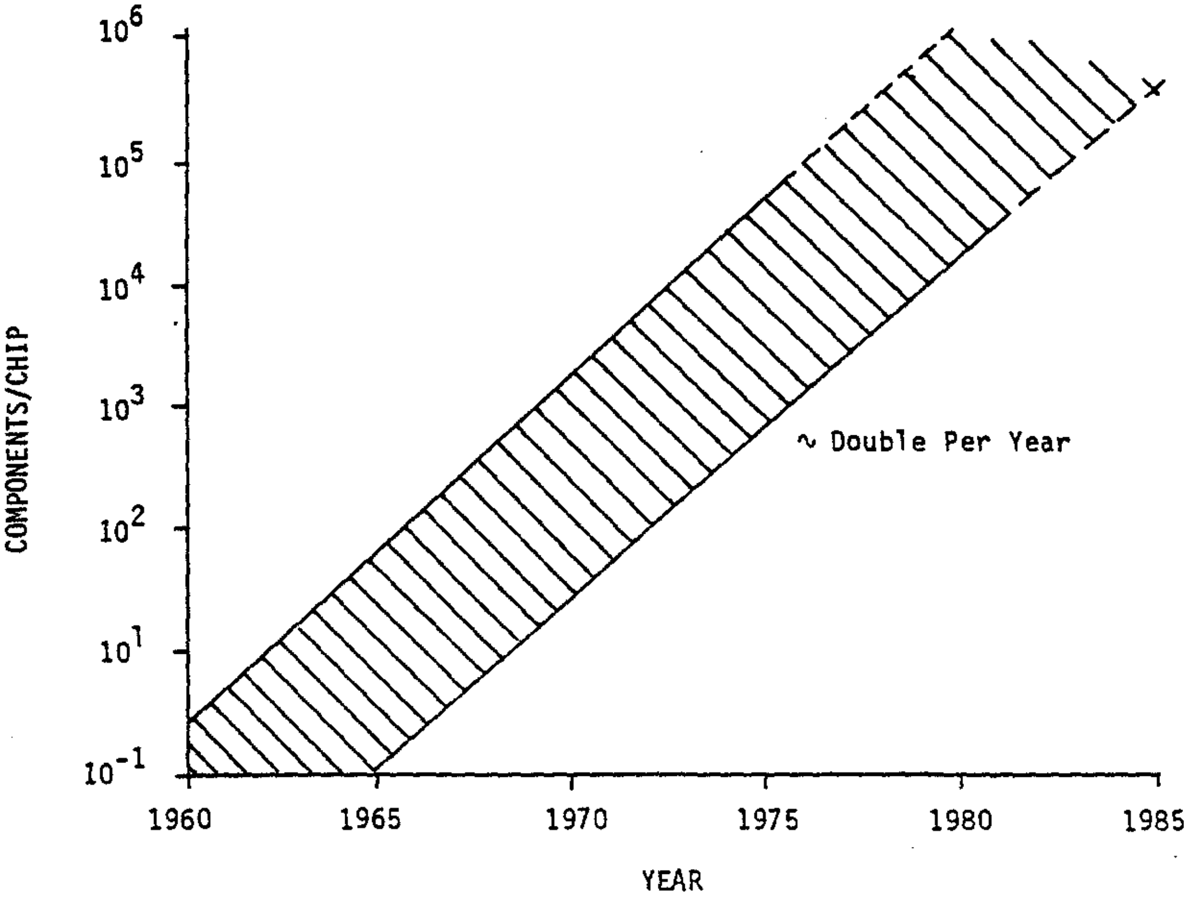


Figure 1.2 The Density Growth of Large Scale Integrated Circuits

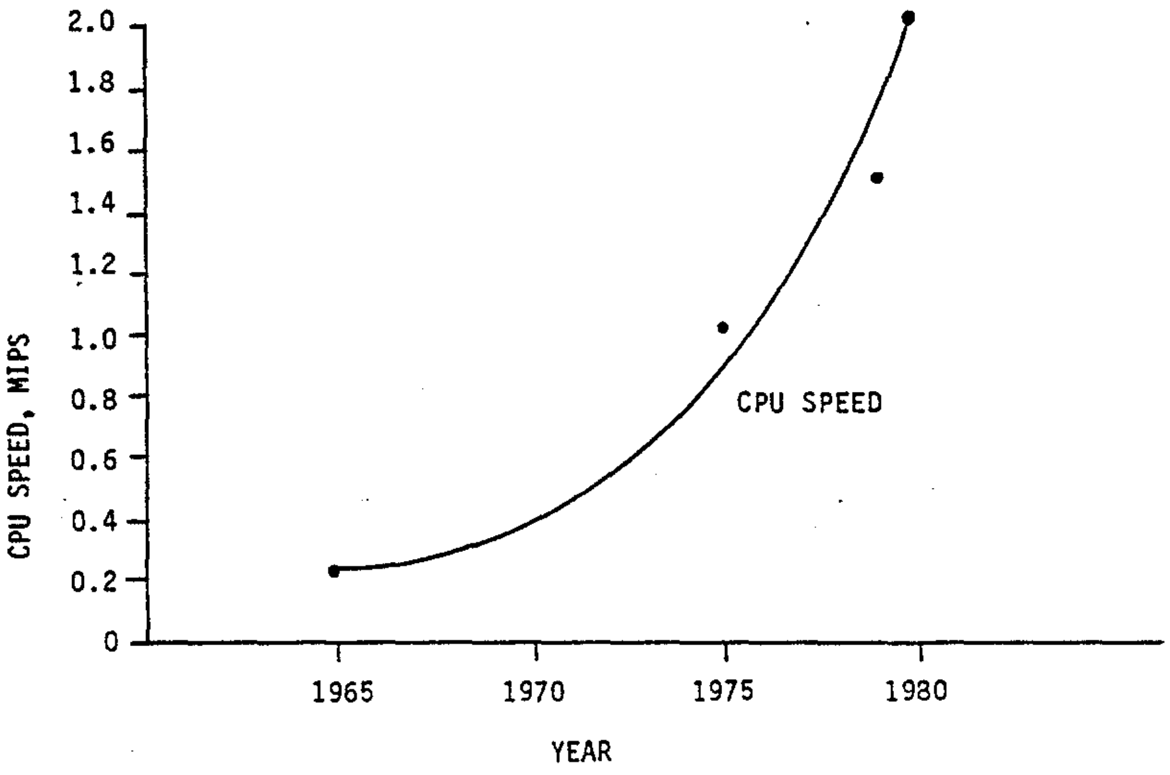


Figure 1.3 The Exponential Growth of CPU Speed

of device manufacturing technologies [MOE78], offer a variety of semiconductor memories with different access times and prices [THE78, UPT78, FET76]. In Table 1.2, the typical access time and power consumption for several semiconductor memory types are shown. Given these diverse types of memories available on the market, the designer must therefore deoide at the design stage the most suitable memory to use. Moreover, magnetic device technologies have also improved significantly. With the improvement of disks, drums and tapes, the invention of the bubble memories [BOB71], and the Electron Beam Access Memories (EBAMs) [HUG75], it is now possible to provide inexpensive secondary and archival storage to the computer system (see Figure 1.4).

With these evolving technologies, there are three significant impacts on the design of computers. First, new technologies add extra design alternatives to the designers which allow the designers to design a system with improved performance and decreased system complexity. An example is shown by the recent developments of bubble memories, CCD memories and EBAMs which have emerged to fill the "access gap" between the two traditional memory technolo-

Table 1.2 Typical values for LSI Semiconductor RAMs (1978) (Price is shown for quantities of 100)

Memory Type	Access Time (nsec)	Power Consumption (mw)	Approx. Price (¢/bit)
16K MOS dynamic	125-300	400-600	0.30
4K NMOS dynamic	150-350	460	0.33
4K ECL static	30	1000	0.85
4K I^2L dynamic	120	450	0.59
4K TTL static	50-70	600-900	0.80-1.00
4K MOS static	55-170	30-500	0.61-0.92
1K CMOS static	150	4	1.02
1K TTL static	40-100	500-800	0.95
1K ECL static	35-60	500-800	1.30

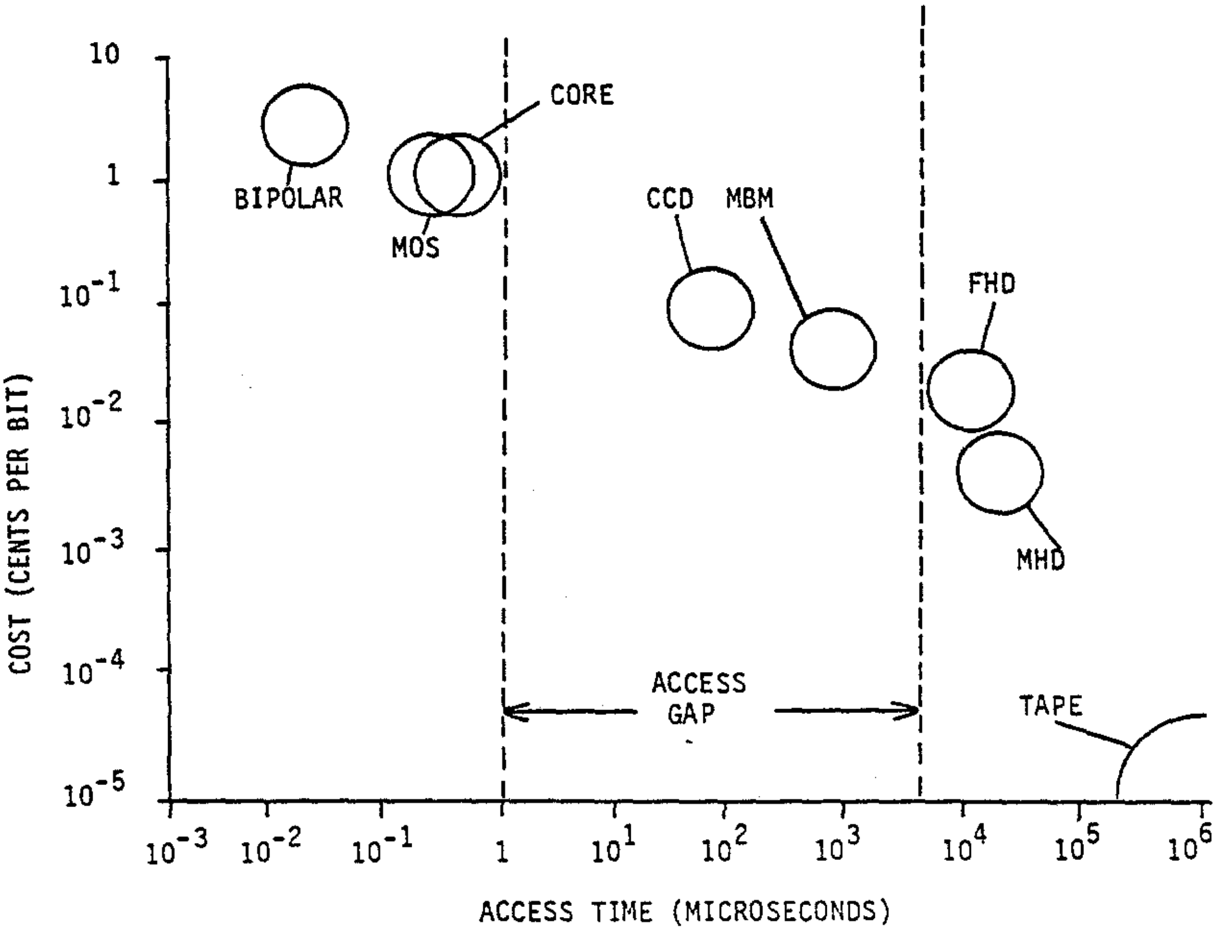


Figure 1.4 Availability of New Memory Technologies

gies (see Figure 1.4). The access gap is the region characterized by an access time between 10^{-6} sec. (MOS memories) and 10^{-3} sec. (fixed head magnetic disk). Much time and effort is expended in finding efficient ways to accomplish at minimum cost the necessary transfers of information across the access gap. With the utilization of "gap-filler" technologies, improved performance and less complex transfer algorithms can be envisioned. Second, increasing logic on a chip allows the designer to incorporate more logical capabilities into the storage sub-system in addition to the storage capabilities. These logical capabilities include abilities to execute arithmetic operations like summation, averaging, as well as logical operations like maximum/minimum searches, equality search, etc. The designer has to decide on the necessary logical capabilities in the system and how they should be designed. The last impact of changing technologies on computer system design is the increasing speed mismatch among the elements of the computer system. With the development of high speed processors such as the CRAY-1 and multi-processor system such as the C.mmp, there is an increasing need of higher bandwidth from the supporting memory sub-system. In order to improve the bandwidths of memories, it is necessary to have intelligent architectural designs and efficient access algorithms for supporting retrieval operations in addition to the utilization of faster memory components. Special emphases should therefore be placed on the utilization of new technologies, the design of new memory architectures and the study of efficient access algorithms.

Evolving technology allows the users more freedom in specifying and operating the system. More stringent requirements can be specified and many of the system's functions can be designed in hardware. However, the dependence of the system on evolving technologies is usually a severe constraint on the designer, and the evolutionary capabilities of a system depend very heavily on how well the designer can predict the future technologies.

(B) *Application Dependence*

Because the size and the complexity of applications change with time, the design of the system may have to be altered after the system has been deployed. However, much too often, systems are designed without taking into account the provision for future changes. When the system evolves, the changes are incorporated into the system in a very disorganized manner. As a result, the unstructureness of the system increases enormously [BEL77] and leads to a regenerative, highly non-linear increase in the effort and cost of the system maintenance [LEH76]. In addition to this, the reliability and the integrity of the system are also jeopardized greatly. One provision is to have a systematic design and development methodology which provides guidelines for the systematic design and construction of DDBs and allows the system to evolve as the application requirements and technology change [RAM78b, RAM79b].

(C) *Standardization*

One of the major inhibiting factors in the development and evolution of DDBs is the lack of standardization in the areas of programming languages, user interface commands, data models, concurrency control mechanisms, hardware components (e.g. disks, tapes), data formats, network protocols, etc. Standardization of hardware and software components allow modular expansion of the system. On the other hand, with a highly evolving technology, standardization may cause costly refitting later and may even hinder acceptance of new ideas.

We have outlined some of the issues in the design of a distributed system supporting a DDB. These issues are by no means complete and other issues, both design and operational, have to be considered. Alternative solutions to these issues provide the options to be decided upon by the designers during the design phase of the system.

1.3 *ARCHITECTURE OF THE SYSTEM SUPPORTING A DDB*

The memory system on a DCS is made up of nodal memories connected together by a network and communicates via the connected processors (Fig. 1.5a). Each node in the system, which consists of a set of processing elements and the supporting storage sub-system, may be active or passive. If the node is active, it acts as a requesting source and can access the memories at other nodes via the communication sub-system. Each of the active nodes in the system has the following functions in addition to the local file accesses.

(1) *Remote access control*

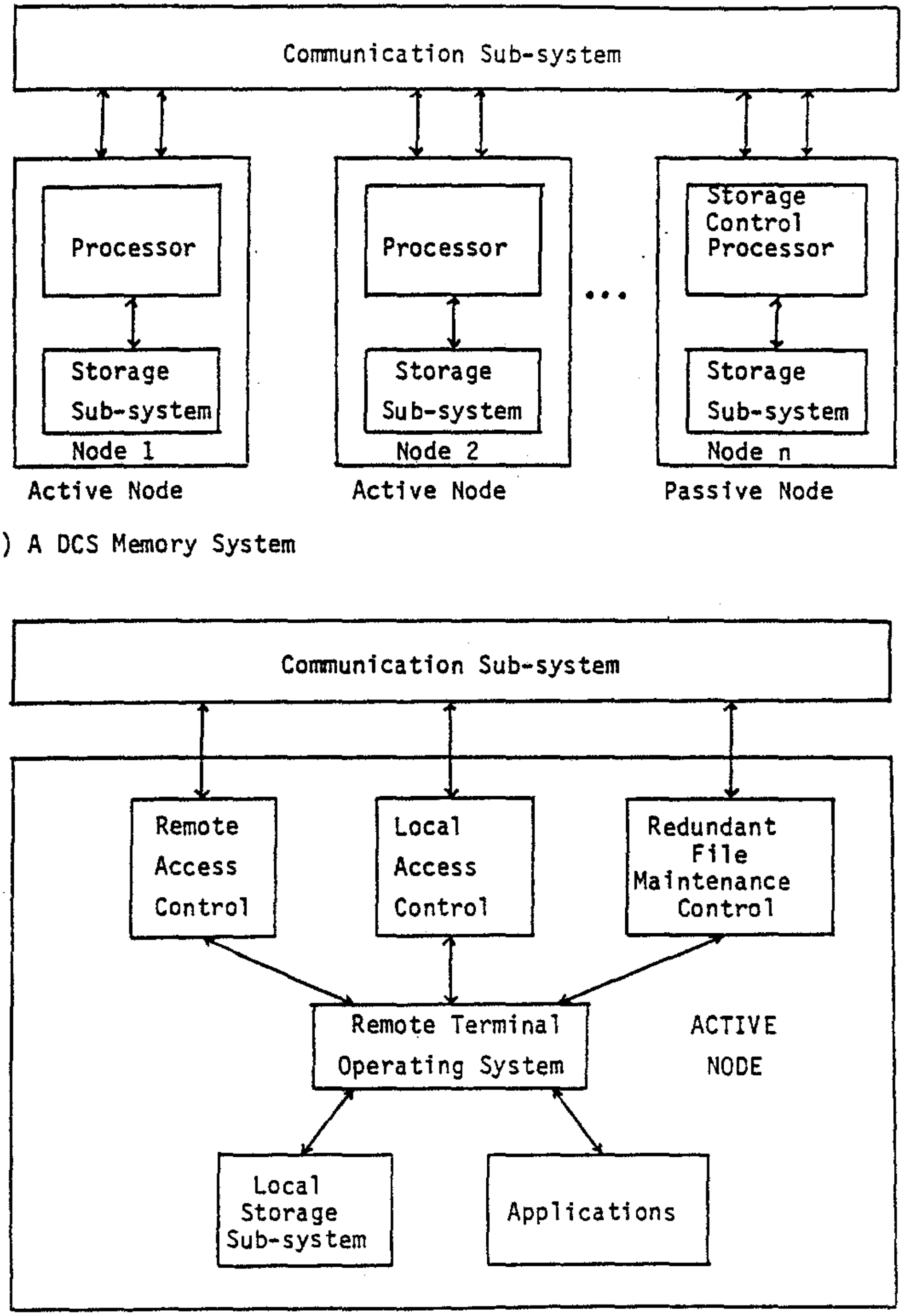
This module detects all remote access requests originating from this node and is responsible for processing them. When a remote request is detected, this module looks up the network directory, and assesses the file status. If the file exists on the network and is accessible by the request, this request will then by transmitted.

(2) *Local access control*

This module is responsible for processing all remote requests received from other nodes in the network. It acts as a security filter and determines whether the file is accessible. If so, the local file is accessed and the data will be transmitted.

(3) *Redundant file maintenance control*

This module coordinates all the local and the remote updates at this node and manages the multiple copies of files on the system. In coordinating updates, if the update originates from a remote node, the status of the file is checked. In case that a conflict occurs and the data cannot be updated, a status message is sent. On the other hand, if no conflict occurs, the file is updated. If the update originates at this node, this module looks up the network directory and sends out all the requested



(a) A DCS Memory System

(b) Functional Design of an Active Node

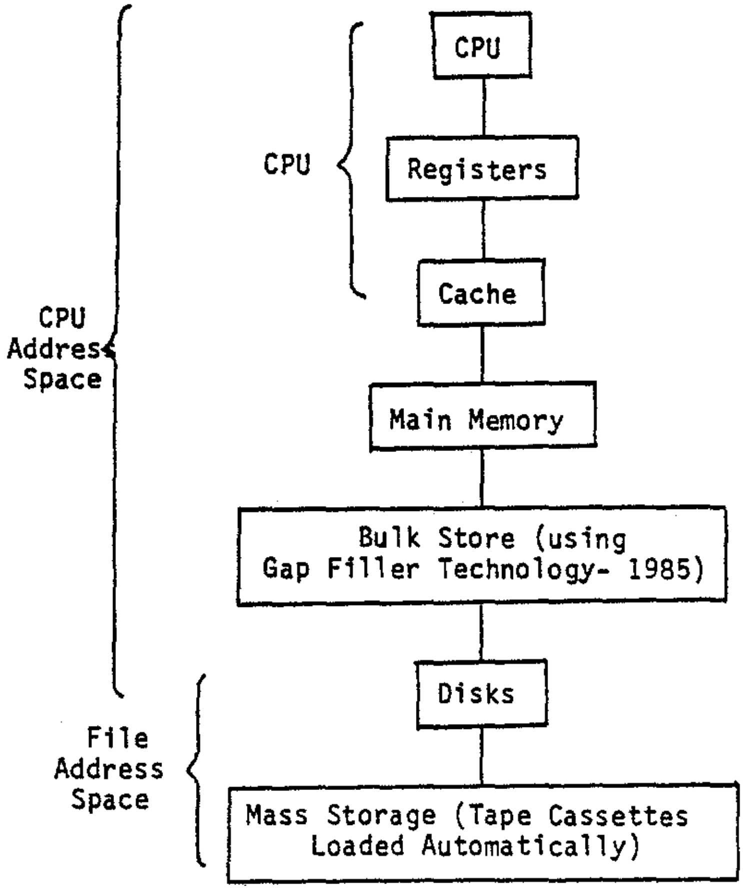
Figure 1.5 Architecture of a DDB System

updates to every redundant copy on the system.

The relation of these modules to each other in an active node is shown in Fig. 1.5b. The logical issues in a DDB, such as security and privacy, concurrency control, etc., are resolved in these modules.

On the other hand, the physical storage system at a node comprises a memory hierarchy that stores programs and data. It has been realized for a long time that the conflicting requirements for high performance and low cost storage sub-system at a node can be satisfied by a combination of expensive high performance devices with inexpensive low performance devices which results in a memory hierarchy. The spectrum of storage devices ranges from bulk store and magnetic tape on one end, to the fast register storage and cache memory in the CPU on the other hand (Figure 1.6). Many issues have to be considered when these different speed elements are put together. These include: the selection of some physical parameters such as the number of levels in the hierarchy and the size and the speed of each level [RAM70, WAR76]; the design of the interconnection mechanism among levels [SMI76, POH75]; the design of efficient scheduling algorithms and record/file distribution and migration algorithms [MUN74, STR77]; the provision of virtual memory support for an automatic file management system [TUE76, POH75, DEN70, BAS70], etc. The last issue is particularly important because the success of a DB is very much dependent on the efficiency of the virtual memory. A file on a DB is likely to be large and cannot reside entirely in the main memory. The use of virtual memory can relieve the users from the laborious task of storage management. It is seen that research is urgently needed in this area.

There is also an increasing tendency to distribute the processing of the CPU to the various levles of the storage sub-system. One successful implementation of this is the DB machine (Figure 1.7). [HSI77]. The DB machine may be a


CPU
Registers
Cache
Main Memory
Bulk Store (using Gap Filler Technology- 1985)
Disks
Mass Storage (Tape Cassettes Loaded Automatically)
CPU
CPU Address Space
File Address Space

	Year 1975	Year 1985
CPU	20 MIPS	40 MIPS
Registers	500B	10^3B
Cache	10^4B	10^6B
Main Memory	10^6B	10^8B
Bulk Store (using Gap Filler Technology- 1985)	0B	10^9B
Disks	10^9B	10^{11}B
Mass Storage (Tape Cassettes Loaded Automatically)	10^{11}B	10^{13}B

Figure 1.6 Storage Hierarchy (With Typical Sizes shown for 1975 and 1985)

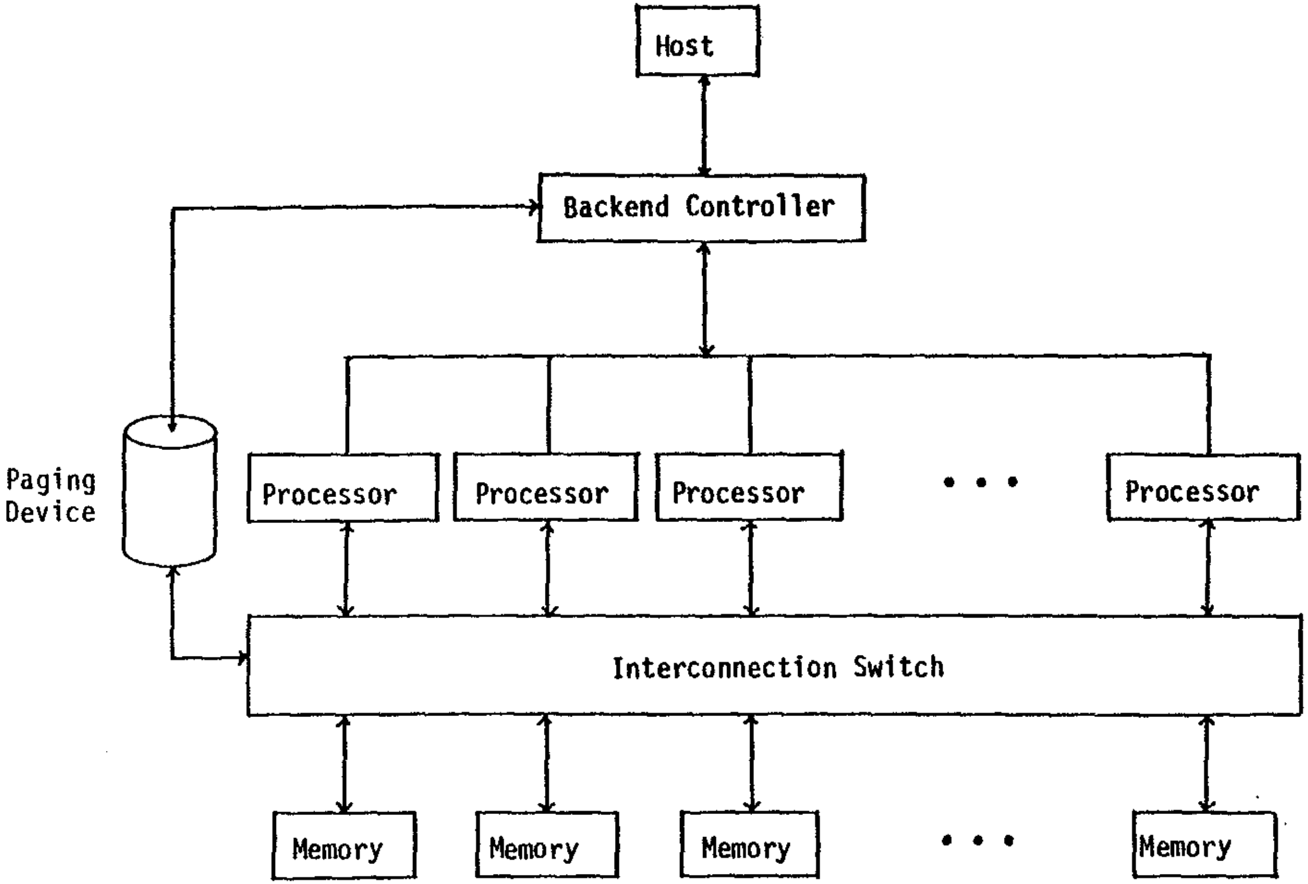


Figure 1.7 Architecture of a Data Base Machine

separate member of the storage sub-system or it may represent a level of the memory hierarchy with additional intelligence. The use of a DB machine relieves the processing load of the central processor and allows more parallelism in the processing of DB requests. Further, processing on large file systems are often I/O bound and many of the file operations are quite simple. A significant communication overhead is incurred in transferring the file to a level of the memory hierarchy where the processor can access it. By distributing the intelligence to the different levels of the memory hierarchy, the DB machine can allow parallel processing with very little communication overhead.

Although DB machines have been successfully designed or implemented, e.g. Data Base Computer (DBC) [BAU76], Context Addressed Segment Sequential Storage (CASSM) [LIP78], Relational Associative Processor (RAP) [OZK77], Rotating Associative Memory for Relational Data Base Applications (RARES) [LIN76], Datacomputer [MAR75], etc., the design of DB machines are still plagued by many issues. Examples of these issues are: deciding on the kind and the degree of parallelism; selecting the appropriate techniques for implementing the storage media; designing the hardware and the software interface; building the storage structure and the backend primitives and designing the control algorithms. These issues are very important because the storage sub-system is very expensive and can be more than 50% of the total hardware system cost [SCH78]. Some of these issues are discussed in Chapter 5 of this thesis.

This section has described some of the necessary architectures in supporting DDB applications. Data base processing generally has some special characteristics and these allow the architecture to be designed differently from conventional architectures. In the next section, the issues on the resource management of data on a DDB are discussed.

1.4 *OBJECTIVES AND CONTRIBUTIONS OF THIS RESEARCH*

1.4.1 *Problem Statement*

The primary objectives of this research effort are the development of a realistic, comprehensive, analytical model for the management of data as resources on a DDB. This design problem encompasses the issues of establishing a systematic way of classification of the different levels of resource management in a DDB, design of performance measures for each level and development of procedures for the optimal solution for certain problems in each level. We hope to provide a strong framework for future research into problems associated with these large scale systems as well as the solutions to some specific design problems.

1.4.2 *Approach*

In order to achieve the global objective, the resource management issues are classified into four related levels, namely, the query level, the file level, the task level and the hardware support level. The specific data management issues investigated are:

(1) *Query Decomposition on DDB's*

A *query* is an access request made by a user or a program in which one or more files have to be accessed. When multiple files are accessed by the same query on a DDB, these files usually have to reside at a common location before the query can be processed. Substantial communication overhead may be involved if these files are geographically distributed and a copy of each file has to be transferred to a common location. It is therefore necessary to decompose the query into sub-queries so that each sub-query accesses a single file. These sub-queries may then be processed in parallel at any location which has a copy of the required file. The results after the processing are sent back to the requesting location. It is generally true that the amount of communications needed to transmit the results is much smaller than the amount needed to transmit the files. This approach has been proposed in the design of the centralized version of INGRES [WON76] and is extended to the design of SDD-1 [WON77], and distributed INGRES [EPS78]. However, in some cases, decomposition is impossible and some file transfers are still necessary. Two techniques are proposed in Chapter 2 so that the overall operational costs of the system can be reduced.

(2) *File Placement and Migration*

This issue relates to the distribution and migration of data base components, namely, files and control programs, on the DDB with the objective of minimizing the overall storage, migration, updating and access costs on the system. A file assignment algorithm is proposed in Chapter 3.

(3) *Task Scheduling*

Requests on the DDB must be scheduled so that high parallelism and overlap can be achieved. The request may be a single word fetch or it may be a page or file access. This parallelism is important because in order to attain high throughput, the parallel hardware and resources must be efficiently utilized. The control of task scheduling can be distributed or centralized. In distributed control, each node may act independently and coordinate with each other. In centralized control, there is a primary node in which all scheduling control are performed. The decision of which is the better control mechanism depends very heavily on the interconnection structure and the communication overhead involved. This issue is discussed in Chapter 4.

(4) *Hardware Support*

In addition to studying the logical data management techniques, the design of the necessary hardware support is also very important. This hardware does not necessary implement a solution to one of the data management issues, e.g. file placement, but it provides auxillary support to these solutions so that they can be implemented efficiently. The particular hardware supports studied are the associative memory and the data base machine. These are discussed in Chapter 5.

The relationships among the various data management issues are shown in Figure 1.8 where a relation → is said to exist between two design issues α, δ, i.e. $\alpha \rightarrow \delta$ if the solution of δ is transparent to the solution of α. That is, the solution of α is not affected by the solution to δ, but not vice versa. The solution to α can therefore be developed independent of δ. In Figure 1.5, it is seen that generally, task scheduling is transparent to file placement and migration which in turn could be transparent to query decomposition. Further, hardware support is transparent to all these logical issues and are generally developed after the algorithms for the logical issues have been designed. Due to the independency, algorithms for query decomposition can be developed independently. In developing algorithms for file placement and migration, the solutions for query decomposition should be taken into account. However, in most cases, assumptions can be made about their solutions and the file placement and migration problem can be solved independently. For example, it may be assumed that all queries which access multiple files may be decomposed into sub-queries that access single files. This assumption is only true in some circumstances, an example of which is shown in Chapter 2 of this thesis. The file placement and migration problem for multiple files is therefore decomposed into many single file optimization sub-problems. It must be noted that other operational control requirements may also impose restrictions on the solutions to the data management issues. For instance, different reliability requirements may demand different lower bounds on the number of copies of a file on the DDB; different

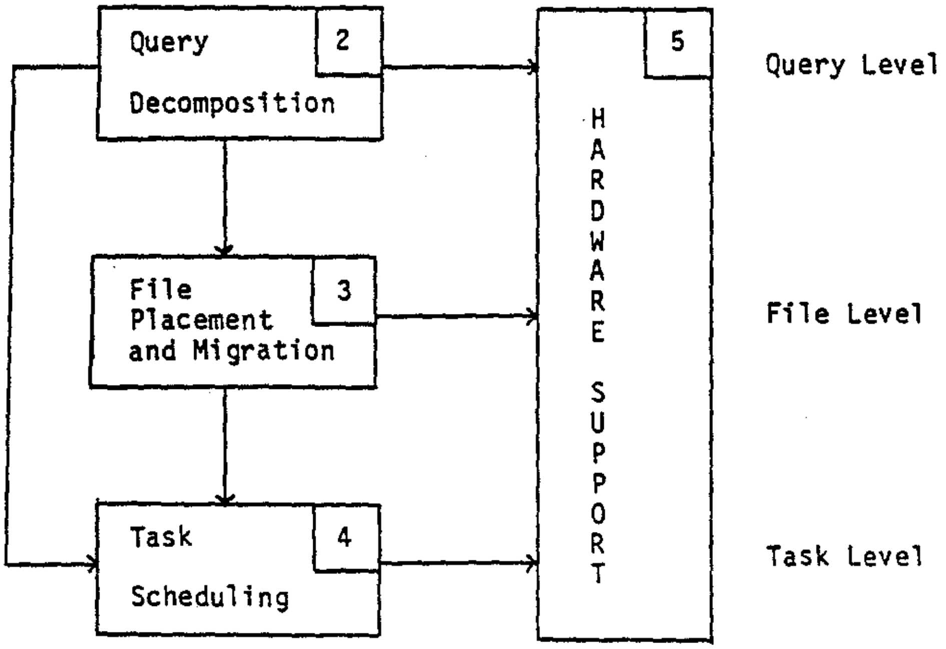


Figure 1.8 Relationships Among Various Data Management Issues (The Number in Each Issue Indicates the Chapter in which it is Discussed)

concurrency control mechanisms may have different costs on the file placement problem; etc. Reasonable assumptions must therefore be made about these techniques in order to determine their effects on the resource management issues and to solve these issues independently.

1.4.3 *Contributions of this Research*

Some specific contributions of this research, arranged in the order of discussion, are listed below.

(A) A model for query decomposition on relational data bases has been developed. It is shown that the optimization of placements of multiple relations can be done independently for each relation.

(B) Two cost reduction models have been designed to reduce the operational costs of a relational data base. The first model reduces the retrieval cost, but increases the update cost. The second model reduces the update cost but increases the retrieval cost. These two cost reduction models can be combined to form a unified approach to reduce the operational costs of the DDB's. Further, it is also shown that the optimization of placements of multiple relations under the use of these techniques can be done independently for each relation.

(C) The isomorphism between the file placement problem and the single commodity warehouse location problem has been proved. Due to this isomorphism, it is also shown that some conditions and techniques developed in computer science to solve the file placement problem are weaker than the corresponding conditions and techniques developed in operations research to solve the warehouse location problem, and vice versa. Further, the technique developed in both problems are inter-changeable.

(D) A file placement heuristic has been developed. While not necessarily yielding optimal system design, this heuristic yields solutions of lower cost than those generated by other currently available heuristics.

(E) A model for the scheduling of tasks on a distributed system has been developed. This model assumes that global control is infeasible and all the scheduling decisions have to be made locally at each node. It is shown that the scheduling of tasks in this model when all the task processing times are deterministic, is an NP-complete problem. A heuristic has been developed and the performance of this heuristic has been verified using simulations.

(F) A more restricted model than the model developed for the scheduling of tasks on a DCS has been proposed. By using the additional constraints, it is shown that the optimal scheduling problem is polynomially solvable. This model actually represents an organization of an interleaved memory system. The performance of the scheduling algorithm has been verified using simulations. Further, the degradation in performance due to dependencies has been estimated.

(G) An associative memory has been designed which is capable of searching the maximum and the minimum in a time independent of the number of words in the memory. It is also capable of doing equality search, threshold searches and proximity search. The design is very efficient and has a complexity of 17 gates per cell. The design is asynchronous and utilizes a word-parallel and bit-serial algorithm. The delay is 1 to 4 gate delays across each bit slice.

(H) The associative memory concept is extended to the design of data base machines. The logic designed can be implemented on the same chip as the memory elements.

2. *QUERY DECOMPOSITION ON A DISTRIBUTED RELATIONAL DATA BASE*

In this chapter, the problems of query decomposition and its association with the optimal placements of relations on a distributed relational DB are studied. Our objectives are to study techniques which allow query decomposition to be done more efficiently and to investigate properties on the optimal placements of multiple copies of relations or segments of relations on the DCS that minimize the total operational cost of the system (e.g. storage cost, multiple update cost, retrieval cost, query processing cost, file migration cost, etc.). The theme of this chapter is to demonstrate that the placements of multiple relations on a distributed relational DB can be optimized for each relation independently. It is assumed that a technique exists to find the optimal placements of multiple copies of a single relation on a DDB, an example for which is shown in Chapter 3. In this chapter, two methods have been proposed to reduce the operational costs of the system. The first method utilizes additional redundant information on the DDB so as to reduce the total retrieval cost and increase the total update cost. The second method uses file partitioning to reduce the total update cost and increase the total retrieval cost. It is shown by an example DB, that under certain conditions, either method, or a combination of both methods, can reduce the total operational costs of the system. A relational data model is chosen in this discussion because it is very popular and the results obtained would be more specific. However, the techniques proposed in this chapter can be generalized to any type of data model and file system.

2.1 *QUERIES ON A RELATIONAL DB*

In a relational DB [COD70], data is viewed as relations of varying degree, the degree being the number of distinct domains participating in the relation. Each instance of a relation is known as a tuple, which has a value for each domain of the relation. Thus a relation can simply be represented in tabular form with columns as domains and rows as tuples.

A *Query* is an access request made by a user or a program, in which one or more relations have to be accessed. A query on a relational DB consists of two parts: the part specifying the domain(s) of the relation to be retrieved and the part specifying the predicate which is a quantification representing the defining properties of the set to be accessed. Let S be a relation of domains s#, sname, city, inventory; and SP be a relation of domains s#, p# (Figure 2.1). The queries on a relational DB can be classified into the following categories [DAT77]:

(1) Retrieval Operations

(a) Single Relation Retrieval: The predicate representing the defining property of the set to be retrieved is defined on the same relation as the set.

(a) Relation S

S	s#	sname	city	inventory
	1	Supplier A	New York	1500
	3	Supplier B	San Francisco	700
	5	Supplier C	Chicago	2500

(b) Relation SP

SP	s#	p#
	1	A1
	2	A1
	3	A2
	4	A2
	5	P2

Figure 2.1 Relations S and SP

E.g. GET (S.sname): (S.city="Paris" AND S.inventory>1000)

(b) Multiple Relation Retrieval: The predicate, as well as the set to be retrieved, may be defined over multiple relations.

E.g. GET (S.sname): (S.s#=SP.s# AND SP.p#="P_2")

Relations S and SP must be available simultaneously before the retrieval can be processed.

(2) Storage Operations

(a) Single Relation Update;

(b) Multiple Relation Update;

(c) Insertion;

(d) Deletion.

(3) Library Functions

These represent more complicated operations on the predicate than the equality operations, e.g. counting the number of occurences, selecting the maximum/minimum etc.

Single relation queries can be processed very easily on a distributed relational DB. When the relation is geographically distributed, the query can be sent to a node that has a copy of the relation and be processed there. The results after the processing can be sent back to the requesting node. It is generally true that the amount of communications needed to transmit the results is much smaller than the amount needed to transmit the relations.

On the other hand, the processing of a mult-relation query is more complicated. When multiple relations are accessed by the same query on a DDB, these relations usually have to reside at a common location before the query can be processed. Substantial communication overhead may be involved if these relations are geographically distributed and a copy of each relation has to be transferred to a common location. It is therefore necessary to decompose the query into sub-queries so that each sub-query accesses a single relation. This technique has been proposed in the design of the centralized version of INGRES [WON76], and is extended to the design of SDD-1 [WON77] and distributed INGRES [EPS78]. Specifically, the technique consists of two steps. The first step is to select a site with the minimum amount of data movements to that site before the query can be processed. This is used as a starting point for the second step of the algorithm which determines the sequence of moves that results in a minimum cost. The algorithm used is a greedy algorithm and only local optima can result from such an algorithm. Hevner and Yao [HEV79] have followed a similar approach and have developed two optimal algorithms for arranging data transmissions and local data processing with minimal response time and minimal total time, for a special class of queries. These optimal algorithms are used as a basis to develop a general query processing algorithm for a general query in which each required relation may have any number of joining domains and output domains and each node may have any number of required relations. This general algorithm is a heuristic which uses an improved exhaustive search to find efficient query distribution strategies. Ghosh also proposed a model of data distribution on a DB which facilitates query processing [GHO76]. Specifically, the model consists of a DB with multiple target segment types and there are queries with multiple target segment types. The objective is to distribute the segments on the DB so as to maximize the number of segments that the queries can retrieve in parallel from different nodes. The model only looks at the problem from a retrieval point of view and no cost is associated with retrieving a segment from a node.

Most of the previous work addresses the problem from two separate viewpoints. The first one is concerned about the questions of what are the processing sequence of the query and where it should be processed. The second viewpoint is concerned about where the files should be placed so that they can be accessed efficiently. These two viewpoints are not entirely independent and should be investigated together. Further, there exists queries which are non-decomposable. For example, the query:

GET (S.sname): (S.s#=SP.s# AND SP.p#="P_2")

is not decomposable into single relation retrievals because there is a logical relation "=" which is defined over a common domain s# of the relations S and SP. These relations must be available simultaneously at a common location before the retrieval or update operations can be performed. Instead of solving the problem of decomposing the queries, we study two techniques to reduce the processing and communication costs for non-decomposable queries in this chapter. It is shown later, by the introduction of some redundant information on the DB and by the use of file partitioning, non-decomposable queries may be made decomposable, (see also [RAM79a, RAM79c]). The basic assumption made over here is that all the required relations are moved to the node at which the query originates, before the processing of the query begins. It is possible to consider a sequence of moves which will minimize the total amount of data transferred. However the problem will be very complicated and the intention of this chapter is to demonstrate the usefulness of the techniques of using redundant information and file partitioning.

Before the techniques are discussed, the problem of placements of relations on a DDB is first formulated.

2.2 *THE PLACEMENTS OF RELATIONS ON A DDB*

In this section, a model for the placements of multiple relations on a DDB is formulated. The model is shown for the special case of two relations and is generalized later to the case of more than two relations.

Consider two relations a and b, the retrieval and the update rates at node i are (see Figure 2.2)[1]:

$q_{i,a}^{a}(q_{i,b}^{b})$ = rate of access at node i for a single relation retrieval accessing relation a(b);

$q_{i,a,b}^{a,b}$ = rate of access at node i for a multi-relation retrieval accessing both relations a and b;

$u_{i,a}^{a}(u_{i,b}^{b})$ = rate of update at node i for a single relation query updating relation a(b);

$u_{i,a}^{a,b}(u_{i,b}^{a,b})$ = rate of update at node i for a multi-relation query accessing both relations a and b before updating relation a(b).

The costs for each unit of access are:

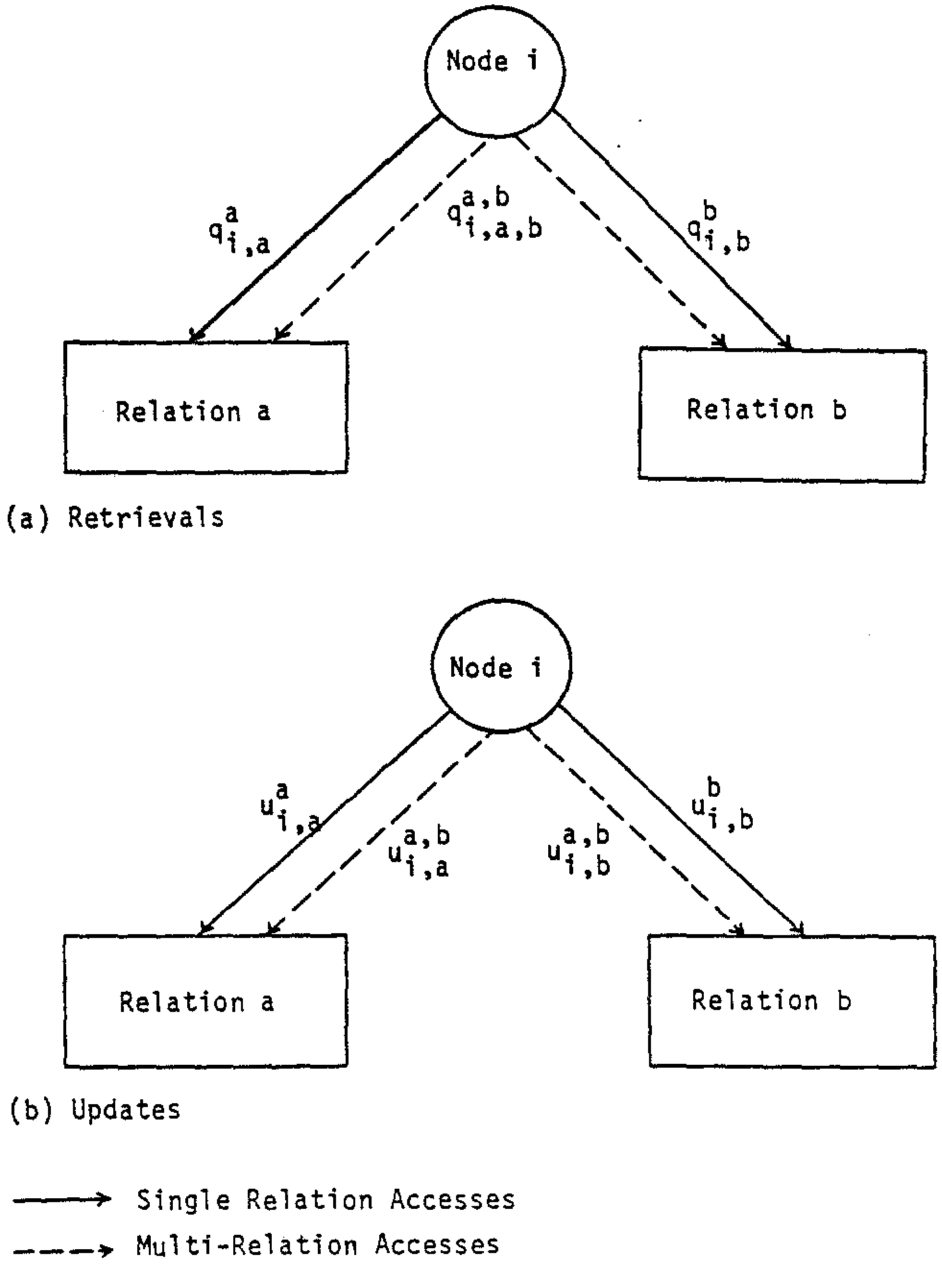
$S_{i,j}^{a}(S_{i,j}^{b})$ = communication and processing cost per unit query of accessing relation a(b) from node i to node j;

$M_{i,j}^{a}(M_{i,j}^{b})$ = communication and processing cost per unit update of multiple updating relation a(b) from node i to node j.

We differentiate between the costs of retrievals and updates because in some applications, retrievals are more important than updates and therefore would have a higher cost (e.g. inventory system); while in other real time applications, updates may be more frequent and therefore more critical (e.g. airline reservation system). Let:

n = number of nodes on the DCS;

[1] The conventions of the symbols used are as follows: i,j represent indexes for nodes; a,b represent indexes for relations; the superscripts represent the list of relations that the query must access before the query can be processed; the subscripts represent the nodes concerned and the tar-



(a) Retrievals

(b) Updates

Figure 2.2 Retrieval and Update Rates on a 2-Relation DDB from Node i

$l_a(l_b)$ = length of relation a(b);

$f_{i,a}(f_{i,b})$ = per unit cost of storing relation a(b) at node i.

We define from the characteristics of the queries initiated from node i, the following symbols:

(1) Single relation retrievals:

$\alpha_{i,a}^{a}(\alpha_{i,b}^{b})$ = fraction of relation a(b) that is put into the result relation due to the execution of a single relation retrieval on a(b);

(2) Multi-relation retrievals:

$\alpha_{i,a}^{a;b}(\alpha_{i,b}^{a;b})$ = fraction of relation a(b) that is needed to process a multi-relation retrieval on a and b;

(3) Single relation updates:

$\beta_{i,a}^{a}(\beta_{i,b}^{b})$ = fraction of relation a(b) that will be updated by a single relation update;

(4) Multi-relation updates:

$\nu_{i,a}^{a;b}(\nu_{i,b}^{a;b})$ = fraction of relation a(b) that is needed to process a multi-relation update before the updates can be performed;

$\beta_{i,a}^{a;b}(\beta_{i,b}^{a;b})$ = fraction of relation a(b) that will be updated by a multi-relation update after relations a and b have been accessed.

In processing a multi-relation update, the relations a and b must be accessed first in order to determine what are the actual updates that have to be made. This is measured by the parameters $\nu_{i,a}^{a;b}$ and $\nu_{i,b}^{a;b}$. The fraction of relations a

get list of relations for the query.

and b to be updated after they have been determined are measured by the parameters $\beta_{i,a}^{a,b}$ and $\beta_{i,b}^{a,b}$.

The parameters defined above can be estimated from the characteristics of the different types of queries that can be made on the DDB and the probability distribution of the data stored in the relations.

The control variables governing the file locations and the routing discipline are defined as follows:

$$Y_i^a(Y_i^b) = \begin{cases} 0 & \text{if } \textit{relation } a(b) \textit{ does not exist at node } i \\ 1 & \textit{otherwise} \end{cases}$$

$X_{i,j}^a(X_{i,j}^b)$ = fraction of queries made at node i on relation a(b) that are routed to node j.

It is true that if $X_{i,j}^r>0$, then $Y_j^r=1$ for r=a,b.

The optimization problem of placing relations a and b on the DDB can be formulated in the following linear program:

min (2.1)

$$\sum_{r=a,b} \sum_{i=1}^{n} \sum_{j=1}^{n} q_{i,r}^r \alpha_{i,r}^r l_r X_{i,j}^r S_{i,j}^r \tag{2.1a}$$

$$+ \sum_{r=a,b} \sum_{i=1}^{n} \sum_{j=1}^{n} q_{i,a,b}^{a,b} \alpha_{i,r}^{a,b} l_r X_{i,j}^r S_{i,j}^r \tag{2.1b}$$

$$+ \sum_{r=a,b} \sum_{i=1}^{n} \sum_{j=1}^{n} u_{i,r}^r \beta_{i,r}^r l_r M_{i,j}^r Y_j^r \tag{2.1c}$$

$$+ \sum_{r=a,b} \sum_{i=1}^{n} \sum_{j=1}^{n} u_{i,r}^{a,b} \Bigg[\sum_{s=a,b} \nu_{i,s}^{a,b} l_s X_{i,j}^s S_{i,j}^s + \beta_{i,r}^{a,b} l_r M_{i,j}^r Y_j^r \Bigg] \tag{2.1d}$$

$$+ \sum_{r=a,b} \sum_{i=1}^{n} f_{i,r} l_r Y_i^r \tag{2.1e}$$

subject to the following constraints:

$$\sum_{i=1}^{n} Y_i^r \geq 1 \qquad r=a,b \tag{2.1f}$$

$$\sum_{j=1}^{n} X_{i,j}^{r} = 1, \qquad r=a,b\,,\ i=1,2,\ldots,n \tag{2.1g}$$

$$nY_j^r \geq \sum_{i=1}^{n} X_{i,j}^{r} \geq 0, \quad r=a,b\,,\ j=1,2,\ldots n \tag{2.1h}$$

$$Y_i^r = 0,1, \qquad r=a,b\,,\ i=1,2,\ldots,n \tag{2.1i}$$

Eq. 2.1a represents the access cost for single relation retrievals; Eq. 2.1b represents the access cost for multi-relation retrievals; Eq. 2.1c represents the update cost for single relation updates; Eq. 2.1d represents the update cost for multi-relation updates and Eq. 2.1e represents the storage cost of relations on the DDB. Condition 2.1f assures that at least one copy of the relation exists; condition 2.1g assures that all the queries are serviced; condition 2.1h assures that the relation must exist at a node if a route is defined to access it at that node and condition 2.1i assures that the control variables Y_i^r are integral.

LEMMA 2.1

The above optimization problem can be partitioned into two independent optimization sub-problems, one for each relation:

(a) min (2.2)

$$\sum_{i=1}^{n} \sum_{j=1}^{n} Q_i^a X_{i,j}^a S_{i,j}^a + \sum_{i=1}^{n} \sum_{j=1}^{n} U_i^a M_{i,j}^a Y_j^a + \sum_{i=1}^{n} F_i^a Y_i^a$$

where

$$Q_i^a = (q_{i,a}^a \alpha_{i,a}^a + q_{i,a,b}^{a,b} \alpha_{i,a}^{a,b} + u_{i,a}^{a,b} \nu_{i,a}^{a,b} + u_{i,b}^{a,b} \nu_{i,a}^{a,b}) l_a$$

$$U_i^a = (u_{i,a}^a \beta_{i,a}^a + u_{i,a}^{a,b} \beta_{i,a}^{a,b}) l_a$$

$$F_i^a = f_{i,a} l_a$$

subject to:

$$\sum_{i=1}^{n} Y_i^a \geq 1$$

$$\sum_{j=1}^{n} X_{i,j}^a = 1 \qquad i=1,\ldots,n$$

$$nY_j^a \geq \sum_{i=1}^{n} X_{i,j}^a \geq 0 \qquad j=1,\ldots,n$$

$$Y_i^a = 0,1 \qquad i=1,\ldots,n$$

(b) min (2.3)

$$\sum_{i=1}^{n}\sum_{j=1}^{n} Q_i^b X_{i,j}^b S_{i,j}^b + \sum_{i=1}^{n}\sum_{j=1}^{n} U_i^b M_{i,j}^b Y_j^b + \sum_{i=1}^{n} F_i^b Y_i^b$$

where

$$Q_i^b = (q_{i,b}^{b}\alpha_{i,b}^{b} + q_{i,a,b}^{a,b}\alpha_{i,b}^{a,b} + u_{i,a}^{a,b}\nu_{i,b}^{a,b} + u_{i,b}^{a,b}\nu_{i,b}^{a,b})l_b$$

$$U_i^b = (u_{i,b}^{b}\beta_{i,b}^{b} + u_{i,b}^{a,b}\beta_{i,b}^{a,b})l_b$$

$$F_{i,b} = f_{i,b}\, l_b$$

subject to:

$$\sum_{i=1}^{n} Y_i^b \geq 1$$

$$\sum_{i=1}^{n} X_{i,j}^b = 1 \qquad i=1,\ldots,n$$

$$nY_j^b \geq \sum_{i=1}^{n} X_{i,j}^b \geq 0 \qquad j=1,\ldots,n$$

$$Y_i^b = 0,1 \qquad i=1,\ldots,n$$

Proof

We notice in optimization problem (2.1) that there are no cross product terms in the control variables of relations a and b. Therefore, the objective function of (2.1) can be written as a sum of objective functions of optimization problems (2.2) and (2.3), and similarly, the constraints can be partitioned into two independent sets. The solution to (2.2) will therefore be a constant in (2.1) which implies that (2.3) can be solved independently. Similarly, the solution to (2.3) will be a constant in (2.1) and this implies that (2.2) can be solved independently.

Q.E.D.

We conclude that the optimization problem 2.1 for relations a and b can be carried out as two optimization sub-problems for relations a and b independently.

A further simplification of the integer programs (2.2) and (2.3) is to first solve for $X_{i,j}^r$, $r=a,b$, and substitute it into the integer programs. It is shown in [ALC76] that,

$$X_{i,j}^r = \begin{cases} 1 & \text{if } S_{i,j}^r = \min_{k, Y_k^r=1} S_{i,k} \\ 0 & otherwise \end{cases}$$

The detailed proof will not be shown here.

A generalization of Lemma 2.1 is to allow any number of relations in the DDB. This is shown in the following theorem.

THEOREM 2.1

The general problem of optimizing the placements of multiple relations on a DDB can be decomposed into multiple sub-problems, one for the placement of each relation.

The proof, which requires some symbols to be defined and can be done by obvious generalization of the proof of Lemma 2.1, will not be shown here.

The importance of Theorem 2.1 is that the original optimization problem of placing multiple copies of m relations on the DDB, which has a complexity of the order of $O(2^{nm})$, is reduced to m simpler optimization sub-problems of placing multiple copies of each relation on the DDB, each of which has a complexity of the order of $O(2^n)$. There are many techniques developed to place multiple copies of a relation on a DDB, e.g. [CAS72, LEV74, MOR77]. Some of these techniques are exhaustive and give optimal solutions, e.g. [CAS72, LEV74, MOR77]; others give sub-optimal solutions and have a polynomial running time, an example of which is shown in Chapter 3 of this thesis. In the remainder of this chapter, we discuss two techniques to minimize the operational costs on the DDB. The costs with and without the application of these techniques are compared.

2.3 *COST REDUCTION ON THE PLACEMENTS OF RELATIONS ON A DDB BY UTILIZING REDUNDANT INFORMATION*

In section 2.1, the technique of query decomposition is briefly described. In query decomposition, optimization is performed on the processing of a single query which originates at a node. The objective is to decompose a multi-relation query into as many single relation sub-queries as possible so that data (relation) movements from one node to another can be minimized. However, there exists non-decomposable queries which require all the relations that they access to be present at a common location. A large number of relation transfers may be needed if these relations are geographically distributed. In order to avoid these extra relation transfers, a technique utilizing redundant information is proposed here. Instead of decomposing queries that access multiple relations, it may be sufficient to provide redundant information in each relation so that multiple relations do not need to reside at a single location before the query can be processed. For example, in processing the query:

GET (S.sname): (S.s#=SP.s# AND SP.p#="P_2")

on two geographically separated relations, S and SP (Figure 2.1), it may be necessary to transfer relation S to the node where SP resides and then process the query there or vice versa. However, if the information (S.s#=SP.s#) is compiled beforehand into the two relations (Figure 2.3), then the above query can be decomposed into two single relation sub-queries:

GET (S.s#, S.sname): (S.s#=SP.s#) and

GET (SP.s#): (S.s#=SP.s# AND SP.p#="P_2").

In this case, the processing can be done in parallel and the amount of

(a) Relation S

S	s#	S.s#= SP.s#	sname	city	inventory
	1	1	Supplier A	New York	1500
	3	1	Supplier B	San Francisco	700
	5	1	Supplier C	Chicago	2500

(b) Relation SP

SP	s#	S.s#= SP.s#	p#
	1	1	A1
	2		A1
	3	1	A2
	4		A2
	5	1	P2

Figure 2.3 Relations S and SP with (S.s#=SP.s#) information compiled into the relations

information transfers is much smaller.

This technique poses several problems. First, it is necessary to take one extra bit for each tuple in order to compile this piece of information. If the amount of information to be added is large, (e.g. when the number of different predicates defined on a common domain of two relations is large), the size of the extra storage space may be significant. Second, when the common domain of one relation is modified, it is necessary to "multiple update" the redundant information in all the common domains of the other relations in the DDB. Referring to Figure 2.3, if an extra tuple with s#="2", sname="Supplier D", city="Boston" and inventory="3000" is added to relation S, then it is necessary to find out what are the changes that have been made on the redundant information (S.s#=SP.s#) in both relations S and SP, and to update these changes in addition to the original update. In this case, the (S.s#=SP.s#) information has to be changed in relations S and SP because relation SP contains a tuple with s#=2. If updating activities are frequent, the "multiple update" cost is large. The net effect of this technique is therefore to reduce the total retrieval cost and to increase the total update cost of the system. Further, the response time in reflecting an update on the DDB may be longer in this case because of the need to update the redundant information. Third, this technique requires that the DB designer be able to estimate the amount of additional information to be compiled into the relations. A possible way is to pre-analyze the type of predicates used in retrievals and updates and to determine what are the essential information to be compiled into the relations. A compromize should be made between introducing extra information with additional storage space and higher cost in multiple updates, and reducing the amount of relation transfers. It would be advantageous for the more frequently used predicates and less advantageous for the others.

In the remainder of this section, a model is developed for deciding how much redundant information is needed on the DDB in order for this technique to be cost effective. We first examine the strategies that have to be used for retrievals and updates.

The strategies on retrievals of a geographically distributed relation is the same as the strategy when no redundant information is used. The necessary information to be used in processing a single relation query is first projected onto temporary files before they are sent to the originating node. In the case of a non-decomposable multi-relation query, all the required relations are sent to the originating node before the query is processed. On the other hand, the strategy on updates is different from the case of no redundant information because it is also necessary to check whether the redundant information is updated.

There are two variations of the update strategy:

(1) The updates are first sent to the multiple copies of the file to be updated;

The necessary information on all the relations, which is needed to determine if the redundant information has to be updated, is sent to a common node;

The updates to be made on the redundant information are determined there;

The updates on the redundant information are sent out to all the affected relations.

(2) The necessary information on all the relations, which is needed to determine if the redundant information has to be updated, is sent to node i where the update originates, (actually, it can be sent to any other node, but the control overhead in doing this would usually be greater);

The update to be made on the redundant information are determined at this node;

The updates on the target relation as well as the updates on the redundant information, are sent out to all the relations.

The advantage of using strategy (1) is that the updates on the target relation are reflected on the DDB in a shorter time than strategy (2). But strategy (1) involves more control overhead and the response time in reflecting the updates on the redundant information is longer than strategy (2). In general, strategy (2) will have a shorter overall response time. We assume that strategy (2) is used in our model.

As before, the model for determining the use of redundant information is first developed for the special case of two relations and is generalized to the case of more than two relations later.

Consider two relations a and b, the retrieval and the update rates, using the notations defined earlier, are shown in Figure 2.4. There are two additional types of single relation retrievals which are decomposed from part of the multi-relation retrievals due to the use of redundant information. In describing the model, the following symbols are defined:

$\gamma_{i,a,b}^{a,b}$ = fraction of non-decomposable multi-relation retrievals on a and b from node i that remain non-decomposable even with the use of redundant information;

$$\frac{\sigma_{i,a}^{a,b}}{(\sigma_{i,a}^{a,b}+\sigma_{i,b}^{a,b})}\left[\frac{\sigma_{i,b}^{a,b}}{(\sigma_{i,a}^{a,b}+\sigma_{i,b}^{a,b})}\right]$$

= fraction of multi-relation-reduced-single-relation retrievals from node i on a(b) due to the use of redundant information;

$(1-\gamma_{i,a,b}^{a,b})q_{i,a,b}^{a,b}$ is the rate of multi-relation retrievals that is decomposable with the use of redundant information;

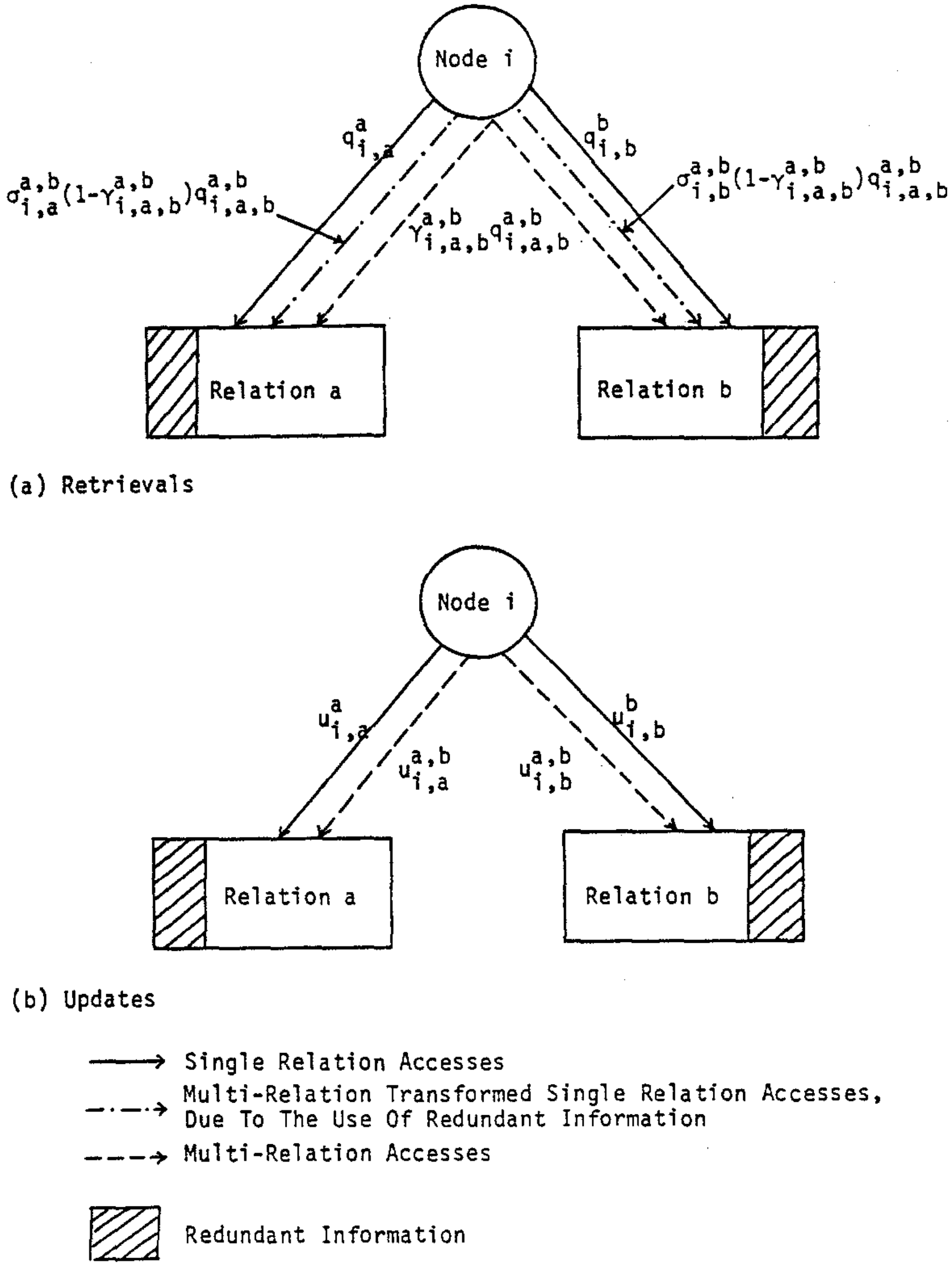
$(1-\gamma_{i,a,b}^{a,b})q_{i,a,b}^{a,b}(\sigma_{i,a}^{a,b}+\sigma_{i,b}^{a,b})$ is the total rate of multi-relation-reduced-single-relation retrievals to relations a and b after the decomposition;

It is generally true that $\sigma_{i,a}^{a,b}+\sigma_{i,b}^{a,b}\geqq 1$, that is, the total rate of additional single relation retrievals after the use of redundant information, is greater than the reduction in multi-relation retrieval rate;

The access rate of multi-relation-reduced-single-relation retrievals on relation r is $(1-\gamma_{i,a,b}^{a,b})q_{i,a,b}^{a,b}\sigma_{i,r}^{a,b}$ for r=a,b;

$\varepsilon_{i,a}^{a,b}(\varepsilon_{i,b}^{a,b})$ = fraction of relation a(b) that is put into the result relation due to a multi-relation-reduced-single-relation retrieval on a(b);

$\delta_{i,a}^{a,b}(\delta_{i,b}^{a,b})$ = fraction of non-decomposable multi-relation updates on a(b) from node i that remain non-decomposable even with the use of


Node i
$q^{a}_{i,a}$
$q^{b}_{i,b}$
$\sigma^{a,b}_{i,a}(1-\gamma^{a,b}_{i,a,b})q^{a,b}_{i,a,b}$
$\sigma^{a,b}_{i,b}(1-\gamma^{a,b}_{i,a,b})q^{a,b}_{i,a,b}$
$\gamma^{a,b}_{i,a,b}q^{a,b}_{i,a,b}$
Relation a
Relation b
Node i
$u^{a}_{i,a}$
$u^{b}_{i,b}$
$u^{a,b}_{i,a}$
$u^{a,b}_{i,b}$
Relation a
Relation b
Single Relation Accesses
Multi-Relation Transformed Single Relation Accesses, Due To The Use Of Redundant Information
Multi-Relation Accesses
Redundant Information

(a) Retrievals

(b) Updates

Figure 2.4 Retrieval And Update Rates On a 2-Relation DDB From Node i Using Additional Redundant Information

redundant information;

$\eta^{a}_{i,a,b}(\eta^{b}_{i,a,b})$ = fraction of updates on relation a(b) from node i that will update redundant information on relations a and b;

$\xi^{a}_{i,a}(\xi^{b}_{i,b})$ = fraction of relation a(b) in which the redundant information has to be updated due to updates originating from node i;

$l'_a(l'_b)$ = length of relation a(b) after the use of redundant information.

In our model, although the amount of storage is greater after redundant information is used, i.e. $l'_r > l_r$ (r=a,b), but the effect on communication is very small because the redundant information does not have to be transferred over the network in processing a query.

The optimization problem of placing relations a and b on the DDB after the use of redundant information can be formulated in the following linear program:

min (2.4)

$$\sum_{r=a,b}\sum_{i=1}^{n}\sum_{j=1}^{n} q^{r}_{i,r}\alpha^{r}_{i,r}l'_{r}X^{r}_{i,j}S^{r}_{i,j} \tag{2.4a}$$

$$+\sum_{r=a,b}\sum_{i=1}^{n}\sum_{j=1}^{n} (1-\gamma^{a,b}_{i,a,b})q^{a,b}_{i,a,b}\sigma^{a,b}_{i,r}\varepsilon^{a,b}_{i,r}l'_{r}X^{r}_{i,j}S^{r}_{i,j} \tag{2.4b}$$

$$+\sum_{r=a,b}\sum_{i=1}^{n}\sum_{j=1}^{n} \gamma^{a,b}_{i,a,b}q^{a,b}_{i,a,b}\alpha^{a,b}_{i,r}l'_{r}X^{r}_{i,j}S^{r}_{i,j} \tag{2.4c}$$

$$+\sum_{r=a,b}\sum_{i=1}^{n}\sum_{j=1}^{n} u^{r}_{i,r}\beta^{r}_{i,r}l'_{r}M^{r}_{i,j}Y^{r}_{j} \tag{2.4d}$$

$$\begin{aligned}+\sum_{r=a,b}\sum_{i=1}^{n}\sum_{j=1}^{n} u^{a,b}_{i,r}\Big[&\delta^{a,b}_{i,r}\sum_{s=a,b}\nu^{a,b}_{i,s}l'_{s}X^{s}_{i,j}S^{s}_{i,j}\\ &+\beta^{a,b}_{i,r}l'_{r}M^{r}_{i,j}Y^{r}_{j}\Big]\end{aligned} \tag{2.4e}$$

$$\begin{aligned}+\sum_{r=a,b}\sum_{i=1}^{n}\sum_{j=1}^{n} \eta^{r}_{i,a,b}(u^{r}_{i,r}+u^{a,b}_{i,r})\Big[&\sum_{s\neq r}\alpha^{a,b}_{i,s}l'_{s}X^{s}_{i,j}S^{s}_{i,j}\\ &+\sum_{t=a,b}\xi^{t}_{i,t}l'_{t}M^{t}_{i,j}Y^{t}_{j}\Big]\end{aligned} \tag{2.4f}$$

$$+ \sum_{r=a,b} \sum_{i=1}^{n} f_{i,r} l'_r Y_i^r \qquad (2.4g)$$

subject to:

$$\sum_{i=1}^{n} Y_i^r \geq 1 \qquad r=a,b$$

$$\sum_{j=1}^{n} X_{i,j}^r = 1 \qquad r=a,b \quad i=1,\ldots,n$$

$$nY_j^r \geq \sum_{i=1}^{n} X_{i,j}^r \geq 0 \qquad r=a,b \quad j=1,\ldots,n$$

$$Y_i^r = 0,1 \qquad r=a,b \quad i=1,\ldots,n$$

Most of the terms in Eq. 2.4 are the same as in Eq. 2.1, except in this case Eq. 2.4b represents the access cost for multi-relation-reduced-single-relation retrievals using the redundant information; and Eq. 2.4f represents the update cost for the redundant information. The term $\eta_{i,a,b}^r(u_{i,r}^r+u_{i,r}^{a,b})$ for r=a,b represents the access rate of updates that may have effects on the redundant information. In determining whether the redundant information will be updated, it is necessary to perform a multi-relation retrieval on the relations concerned. In this case, since we know the updates to be made on relation r, we can fetch a copy of all other relations $s \neq r$ and move the copies to node i. This cost is represented by the term $\sum_{s \neq r} \alpha_{i,s}^{a,b} l'_s X_{i,j}^s S_{i,j}^s$ in Eq. 2.4f. After the updates on the redundant information have been determined, the actual updates, together with the updates on the redundant information are sent to all the nodes which have a copy of the relation. This cost is represented by the term $\sum_{t=a,b} \xi_{i,t}^t l'_t M_{i,j}^t Y_j^t$ in Eq. 2.4f.

A similar lemma and theorem can be proved for this problem.

LEMMA 2.2

Optimization problem 2.4 can be partitioned into two independent optimization sub-problems, one for each relation:

$$(a)\ \min \sum_{i=1}^{n} Q_i^a \min_{j,Y_j^a=1} S_{i,j}^a + \sum_{i=1}^{n}\sum_{j=1}^{n} U_i^a M_{i,j}^a Y_j^a + \sum_{i=1}^{n} F_i^a Y_i^a \tag{2.5}$$

where:

$$Q_i^a = [q_{i,a}^a \alpha_{i,a}^a + (1-\gamma_{i,a,b}^{a,b}) q_{i,a,b}^{a,b} \sigma_{i,a}^{a,b} \varepsilon_{i,a}^{a,b} + \gamma_{i,a,b}^{a,b} q_{i,a,b}^{a,b} \alpha_{i,a}^{a,b}$$
$$+ u_{i,a}^{a,b} \delta_{i,a}^{a,b} \nu_{i,a}^{a,b} + u_{i,b}^{a,b} \delta_{i,b}^{a,b} \nu_{i,a}^{a,b} + \eta_{i,a,b}^{b} (u_{i,b}^{b} + u_{i,b}^{a,b}) \alpha_{i,a}^{a,b}] l'_a$$

$$U_i^a = [u_{i,a}^a \beta_{i,a}^a + u_{i,a}^{a,b} \beta_{i,a}^{a,b} + \eta_{i,a,b}^{a} (u_{i,a}^a + u_{i,a}^{a,b}) \xi_{i,a}^a$$
$$+ \eta_{i,a,b}^{b} (u_{i,b}^{b} + u_{i,b}^{a,b}) \xi_{i,a}^{a}] l'_a$$

$$F_i^a = f_{i,a} l'_a$$

subject to:

$$\sum_{i=1}^{n} Y_i^a \geqq 1$$

$$Y_i^a = 0,1 \qquad i=1,\ldots,n$$

$$(b)\ \min \sum_{i=1}^{n} Q_i^b \min_{j,Y_j^b=1} S_{i,j}^b + \sum_{i=1}^{n}\sum_{j=1}^{n} U_i^b M_{i,j}^b Y_j^b + \sum_{i=1}^{n} F_i^b Y_i^b \tag{2.6}$$

where:

$$Q_i^b = [q_{i,b}^b \alpha_{i,b}^b + (1-\gamma_{i,a,b}^{a,b}) q_{i,a,b}^{a,b} \sigma_{i,b}^{a,b} \varepsilon_{i,b}^{a,b} + \gamma_{i,a,b}^{a,b} q_{i,a,b}^{a,b} \alpha_{i,b}^{a,b}$$
$$+ u_{i,a}^{a,b} \delta_{i,a}^{a,b} \nu_{i,b}^{a,b} + u_{i,b}^{b} \delta_{i,b}^{a,b} \nu_{i,b}^{a,b} + \eta_{i,a,b}^{a} (u_{i,a}^{a} + u_{i,a}^{a,b}) \alpha_{i,b}^{a,b}] l'_b$$

$$U_i^b = [u_{i,b}^b \beta_{i,b}^b + u_{i,b}^{a,b} \beta_{i,b}^{a,b} + \eta_{i,a,b}^{a} (u_{i,a}^a + u_{i,a}^{a,b}) \xi_{i,b}^b$$
$$+ \eta_{i,a,b}^{b} (u_{i,b}^{b} + u_{i,b}^{a,b}) \xi_{i,b}^{b}] l'_b$$

$$F_i^b = f_{i,b} l'_b$$

subject to:

$$\sum_{i=1}^{n} Y_i^b \geq 1$$

$$Y_i^b = 0,1 \qquad i=1,\ldots,n$$

THEOREM 2.2

The general problem of optimizing the placements of multiple relations on a DDB using additional redundant information can be decomposed into multiple sub-problems, one for the placement of each relation.

The proofs of Lemma 2.2 and Theorem 2.2 are very similar to that of Lemma 2.1 and Theorem 2.1 and will not be illustrated here.

We demonstrate the use of this technique in the next section with a simple example.

2.4 *A NUMERICAL EXAMPLE TO ILLUSTRATE THE USE OF REDUNDANT INFORMATION ON A DDB*

In this section, we show by the use of a numerical example, the cost improvement when redundant information is introduced on the DDB.

Consider a DCS of 3 nodes with two relations, S and SP, on the DDB. Let S has domains s#(1), sname(10), city(5), inventory(2) and SP has domains s#(1), p#(1)[2]. Assume that S has 500 tuples and SP has 10000 tuples. The following parameters are also assumed:

$$[S_{i,j}] = [M_{i,j}] = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1.5 \\ 2 & 1.5 & 0 \end{bmatrix} * 10^{-3}$$

$$f_{i,S} = f_{i,SP} = 0$$

$l_S = l'_S = 500{*}18 = 9000$ (words)[3]

$l_{SP} = l'_{SP} = 10000{*}2 = 20000$ (words)[3]

Node	Parameters						
i	$q^S_{i,S}$	$u^S_{i,S}$	$u^{S,SP}_{i,S}$	$q^{SP}_{i,SP}$	$u^{SP}_{i,SP}$	$u^{S,SP}_{i,SP}$	$q^{S,SP}_{i,S,SP}$
1	100	20	115	80	120	40	100
2	50	100	50	100	25	35	50
3	75	15	35	50	15	10	75

and for all iε\{1,2,3\},

$$\alpha^S_{i,S} = \alpha^{SP}_{i,SP} = \nu^{S,SP}_{i,S} = \nu^{S,SP}_{i,SP} = 0.1$$

[2] The number in the parenthesis indicates the length in words in each domain.

[3] Note that $l_r = l'_r$ (r=S,SP) because in this case, we do not consider the cost of storage on the DDB ($f_{i,r}=0$, $r=S,SP$) and the redundant information usually does not have to be sent over the

$$\varepsilon_{i,S}^{S,SP} = \varepsilon_{i,SP}^{S,SP} = 0.05$$

$$\alpha_{i,S}^{S,SP} = \alpha_{i,SP}^{S,SP} = 0.3$$

$$\beta_{i,S}^{S} = \beta_{i,SP}^{SP} = \beta_{i,S}^{S,SP} = \beta_{i,SP}^{S,SP} = 0.25$$

$$\sigma_{i,S}^{S,SP} = \sigma_{i,SP}^{S,SP} = 0.6$$

$$\xi_{i,S}^{S} = \xi_{i,SP}^{SP} = 0.05$$

These parameters have been chosen based on some estimated distribution of the data stored in the relations and the characteristics of the queries made on these two relations. They have been set independent of the nodes and the relations for easy understanding. The fixed cost of storage on the system have all been neglected because the storage cost is usually very small as compared to the communication cost. It is intended to show by this example, the amount of redundant information needed in order for this technique to be cost effective.

In Figures 2.5 and 2.6, two graphs are plotted to show the ratio of cost with redundancy and cost without redundancy against $\delta_{i,r}^{S,SP}$ [4]. In Figure 2.5, the graph is plotted for various values of $\gamma_{i,S,SP}^{S,SP}$ [4] , with $\eta_{i,S,SP}^{r}$ [4] fixed at 0.5. Similarly, in Figure 2.6, the graph is plotted for various values of $\eta_{i,S,SP}^{r}$ [4], with $\gamma_{i,S,SP}^{S,SP}$ [4] fixed at 0.5. It is seen from these two graphs that whenever sufficient redundant information is added to the DDB so that over half of the non-decomposable queries or updates become decomposable, the resultant operational costs are less than the costs without the use of redundant information. Further, it is seen from Figure 2.6 that when the fraction of updates that will update the redundant information is less than 0.5, there is, in general, a cost improvement.

The results we have shown in the example are for illustration. More detailed evaluations are necessary before any definite conclusions can be drawn.

network in order to process a query.

[4] It is assumed that r=S, SP; the variables $\delta_{i,r}^{S,SP}$, $\eta_{i,S,SP}^{r}$ are independent of i and r and the variables $\gamma_{i,S,SP}^{S,SP}$ are independent of r.

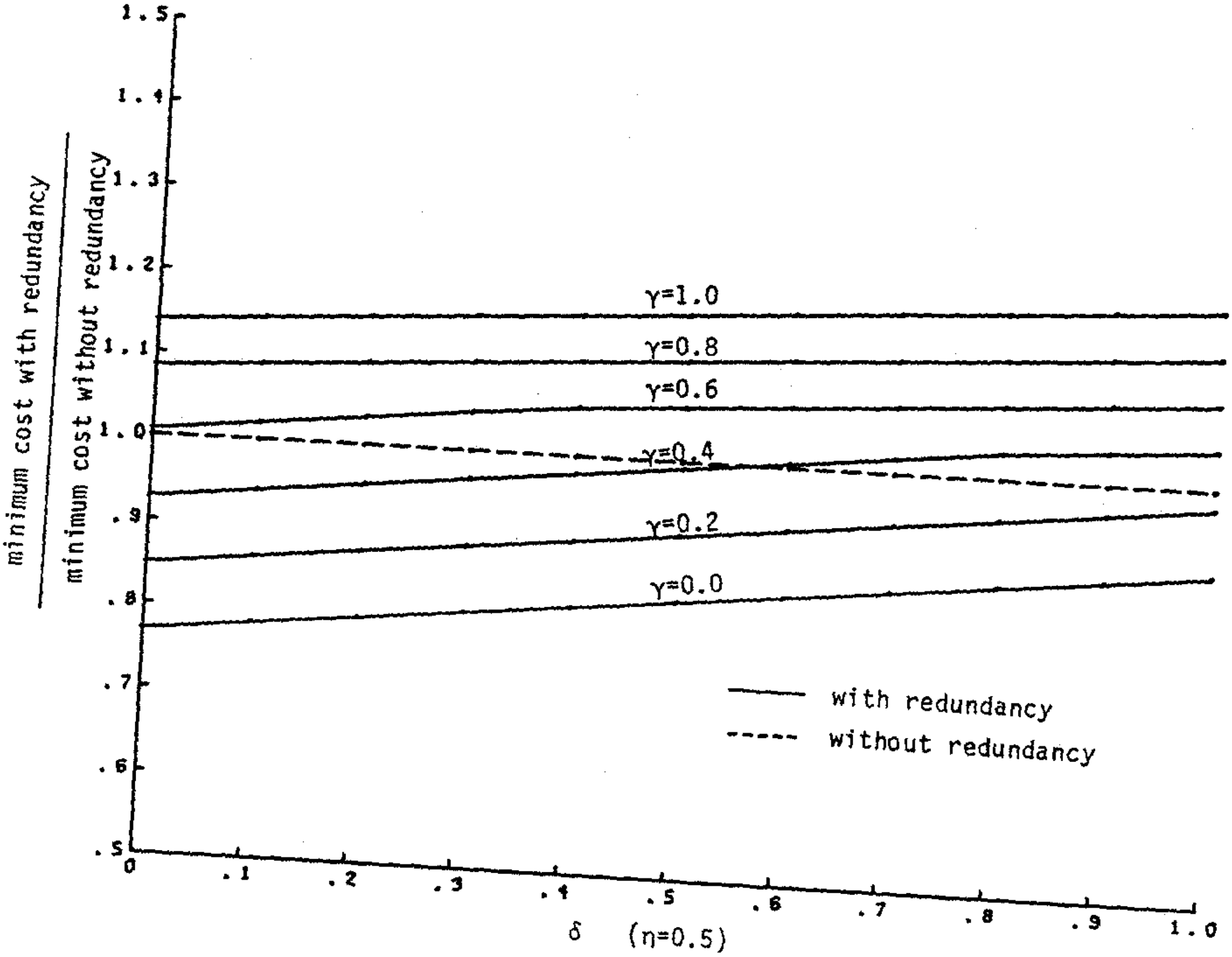


Figure 2.5 A Plot of Cost Ratio with respect to γ for various values of δ under the use of redundant information (it is assumed that γ, δ and η are independent of r=S, SP and i)

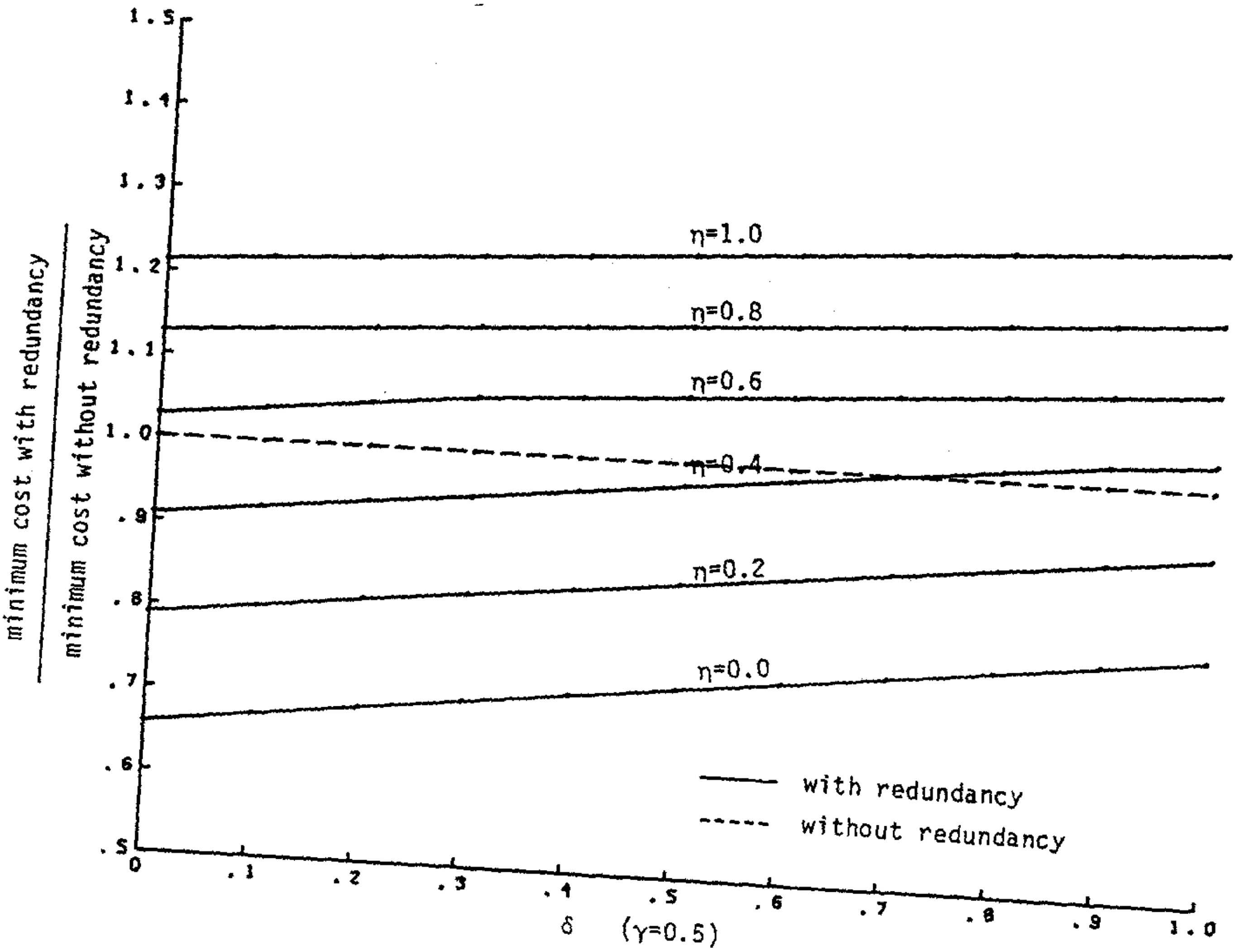
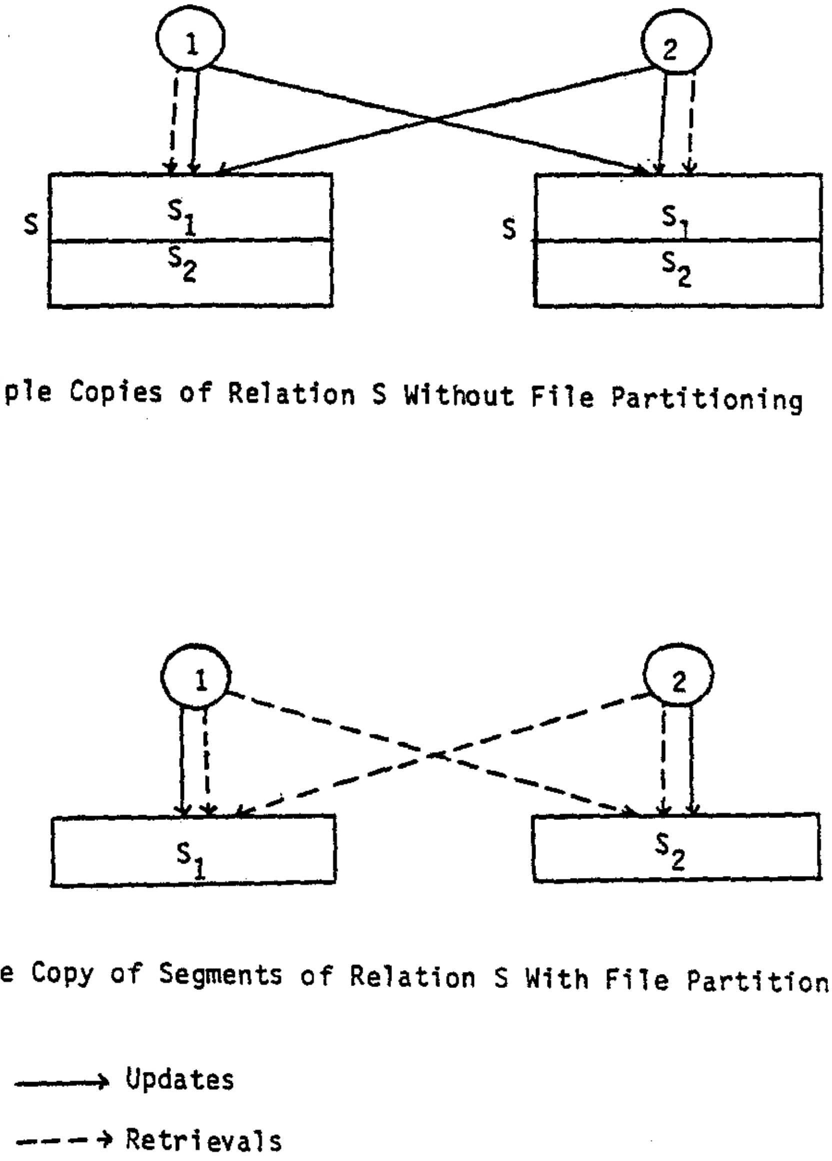


Figure 2.6 A Plot of Cost Ratio with respect to η for various values of δ under the use of redundant information (It is assumed that γ, δ and η are independent of r=S, SP and i)

2.5 *COST REDUCTION ON THE PLACEMENTS OF RELATIONS ON A DDB BY FILE PARTITIONING*

In section 2.3 we have shown a technique by which the total operational costs can be reduced by decreasing the total retrieval cost and increasing the total update cost. We study in this section, the dual of the previous technique, that is, a technique by which the total operational costs can be reduced by decreasing the total update cost and increasing the total retrieval cost. Before the technique is described, the characteristics of an update is first studied.

An update on a relation can broadly be divided into two types. The first type updates only a small segment of the relation and the second type updates all the tuples in the relation. As an example, consider an employee relation. The first type can be an update which increases the salary of a particular employee and the second type can be an update which increases the salary of all the employees in the relation. If the first type is more prevalent, and there is a locality of the updates on the DDB, then the total update cost can reduced by partitioning the relation into segments and distributing the segments to the various nodes of the DDB instead of distributing multiples copies of the relation to the various nodes. On the other hand, the entire relation usually has to be accessed in a retrieval or in a multi-relation update in which the target information to be updated must first be determined. The relation must be searched tuple by tuple in order to determine the set of tuples satisfying the predicate. If a relation is partitioned and distributed on the DDB, all the segments have to be assembled before the retrieval can be made. This cost is likely to be greater than the cost of accessing a copy of the entire relation on the DDB. The resultant cost of file partitioning is therefore an increase in the total retrieval cost and a decrease in the total update cost. The use of file partitioning is further illustrated in Figure 2.7.



(a) Multiple Copies of Relation S Without File Partitioning

(b) Single Copy of Segments of Relation S With File Partitioning

Figure 2.7 The Retrievals and Updates on a DDB (2 Nodes) With and Without File Partitioning

The problems that are related to file partitioning are two folds: how to partition the relations and after the relations are partitioned, how to distribute the segments on the DDB. The first problem can be solved by studying the characteristics of the updates made at different nodes of the DCS and partitioning the relation according to these characteristics. There exist algorithms to solve this problem, e.g., by clustering [JAR71, BON64]. We are therefore more concerned with the problem of distributing the segments of the relations on the DDB after they have been partitioned. In this section, the case with no extra redundant information is first considered and the case with additional redundant information is considered in section 2.7. The model developed here is shown for the special case of two relations and is generalized later to the case of more than two relations.

In addition to the symbols defined in section 2.2, we define the following symbols here. Let

P_a (P_b) = number of segments that relation a (b) is partitioned into;

a_j (b_j) = the j'th segment of relation a (b), j = 1, ..., P_a (P_b);

P = $\{a_1, a_2, \ldots, a_{P_a}\} \cup \{b_1, b_2, \ldots, b_{P_b}\}$.

For single relation queries,

$fr(a_j \,|\, q^a_{i,a})$ $[fr(b_j \,|\, q^b_{i,b})]$ = fraction of retrievals accessing the j'th segment of relation a (b) given that the retrieval rate is $q^a_{i,a}$ $(q^b_{i,b})$;

$fu(a_j \,|\, u^a_{i,a})$ $[fu(b_j \,|\, u^b_{i,b})]$ = fraction of updates on the j'th segment of relation a (b) given that the update rate is $u^a_{i,a}$ $(u^b_{i,b})$;

For multi-relation queries,

$fr(a_j \,|\, q^{a,b}_{i,a,b})$ $[fr(b_j \,|\, q^{a,b}_{i,a,b,})]$ = fraction of multi-relation retrievals accessing the j'th segment of relation a(b) given that the retrieval rate is $q^{a,b}_{i,a,b}$;

$fr(t_j \,|\, u^{a,b}_{i,a})$ $[fr(t_j \,|\, u^{a,b}_{i,b})]$ = fraction of multi-relation updates that have to access the j'th segment of relation t (t = a,b) in order to determine the actual updates, given that the update rate is $u_{i,a}^{a,b}(u_{i,b}^{a,b})$;

$fu(a_j \,|\, u_{i,a}^{a,b})$ $[fu(b_j \,|\, u_{i,b}^{a,b})]$ = fraction of multi-relation updates on the j'th segment of relation a (b) given that the update rate is $u_{i,a}^{a,b}(u_{i,b}^{a,b})$;

It is further assumed that the parameters α, β, γ, f are independent of the effects of partitioning. The optimization problem of placing P_a segments of relation a and P_b segments of relation b on the DDB can be formulated in the following linear program.

min (2.7)

$$\sum_{s=a,b} \sum_{k=1}^{P_s} \sum_{i=1}^{n} \sum_{j=1}^{n} q_{i,s}^{s} fr(s_k \,|\, q_{i,s}^{s}) \alpha_{i,s}^{s} l_{s_k} X_{i,j}^{s_k} S_{i,j}^{s} \tag{2.7a}$$

$$+ \sum_{s=a,b} \sum_{k=1}^{P_s} \sum_{i=1}^{n} \sum_{j=1}^{n} q_{i,a,b}^{a,b} fr(s_k \,|\, q_{i,a,b}^{a,b}) \alpha_{i,s}^{a,b} l_{s_k} X_{i,j}^{s_k} S_{i,j}^{s} \tag{2.7b}$$

$$+ \sum_{s=a,b} \sum_{k=1}^{P_s} \sum_{i=1}^{n} \sum_{j=1}^{n} u_{i,s}^{s} fu(s_k \,|\, u_{i,s}^{s}) \beta_{i,s}^{s} l_{s_k} M_{i,j}^{s} Y_j^{s_k} \tag{2.7c}$$

$$\begin{aligned} + \sum_{s=a,b} \sum_{k=1}^{P_s} \sum_{i=1}^{n} \sum_{j=1}^{n} u_{i,s}^{a,b} \Bigg[& \sum_{t=a,b} \sum_{e=1}^{P_t} fr(t_e \,|\, u_{i,s}^{a,b}) \nu_{i,t}^{a,b} l_{t_e} X_{i,j}^{t_e} S_{i,j}^{t} \\ & + fu(s_k \,|\, u_{i,s}^{a,b}) \beta_{i,s}^{a,b} l_{s_k} M_{i,j}^{s} Y_j^{s_k} \Bigg] \end{aligned} \tag{2.7d}$$

$$+ \sum_{s=a,b} \sum_{k=1}^{P_s} \sum_{i=1}^{n} f_{i,s} l_{s_k} Y_i^{s_k} \tag{2.7e}$$

subject to

$$\sum_{i=1}^{n} Y_i^p \geqq 1 \tag{2.7f}$$

$$\sum_{j=1}^{n} X_{i,j}^p = 1 \tag{2.7g}$$

$$nY_j^p \geqq \sum_{i=1}^{n} X_{i,j}^p \geqq 0 \tag{2.7h}$$

$$Y_i^p = 0,\ 1 \tag{2.7i}$$

$$1 \leqq \sum_{j=1}^{P_s} fr(s_j \,|\, q_{i,s}^{s}) \leqq P_s \tag{2.7j}$$

$$1 \leqq \sum_{j=1}^{P_s} fr(s_j \,|\, q_{i,a,b}^{a,b}) \leqq P_s \tag{2.7k}$$

$$1 \leqq \sum_{j=1}^{P_s} fu(s_j \,|\, u_{i,s}^{s}) \leqq P_s \tag{2.7l}$$

$$1 \leqq \sum_{j=1}^{P_s} fr(s_j \,|\, u_{i,t}^{a,b}) \leqq P_s \tag{2.7m}$$

$$1 \leqq \sum_{j=1}^{n} fu(s_j \,|\, u_{i,s}^{a,b}) \leqq P_s \tag{2.7n}$$

where s,t ∈ {a,b}, i,j ∈ {1,2,...,n} and $p \in \{a_1, \ . \ . \ . \ , \ a_{P_a}\}$ $(cu\{b_1, \ . \ . \ . \ , \ b_{P_b}\}$

Eq. 2.7a to Eq. 2.7i are similar to the corresponding equations in Eq. 2. Eq. 2.7j to Eq. 2.7n represent the conditions that one or more of the segments may have to be accessed when a relation is queried. A lemma and theorem similar to Lemma 2.1 and Theorem 2.1 can be proved for this problem.

LEMMA 2.3

Optimization problem 2.7 can be partitioned into P_a+P_b independent optimization sub-problems, one for each segment. The optimization sub-problem for segment s_k where $s \in \{a, b\}$, $k \in \{1, ..., P_s\}$ is:

$$\min \sum_{i=1}^{n} Q_i^{s_k} \min_{j, Y_j^{s_k}=1} S_{i,j}^{s} + \sum_{i=1}^{n} \sum_{j=1}^{n} U_i^{s_k} M_{i,j}^{s} Y_j^{s_k} + \sum_{i=1}^{n} F_i^{s} Y_i^{s_k} \tag{2.8}$$

where

$$Q_i^{s_k} = \Big[q_{i,s}^{s} fr(s_k \,|\, q_{i,s}^{s}) \alpha_{i,s}^{s} + q_{i,a,b}^{a,b} fr(s_k \,|\, q_{i,a,b}^{a,b}) \alpha_{i,s}^{a,b} + \sum_{t=a,b} u_{i,t}^{a,b} fr(s_k \,|\, u_{i,t}^{a,b}) \nu_{i,s}^{a,b} \Big] l_{s_k}$$

$$U_i^{s_k} = \Big[u_{i,s}^{s} fu(s_k \,|\, u_{i,s}^{s}) \beta_{i,s}^{s} + u_{i,s}^{a,b} fu(s_k \,|\, u_{i,s}^{a,b}) \beta_{i,s}^{a,b} \Big] l_{s_k}$$

$$F_i^{s_k} = f_{i,s} l_{s_k}$$

subject to

$$\sum_{i=1}^{n} Y_i^{s_k} \geqq 1$$

$Y_i^{s_k} = 0, 1 \qquad i=1, \ldots, n$

A generalization of Lemma 2.3 is to allow any number of relations in the DDB. This is shown in the following theorem.

THEOREM 2.3

The general problem of optimizing the placements of multiple relations on a DDB using file partitioning can be decomposed into multiple sub-problems, one for the placement of each partition independently.

The proofs of Lemma 2.3 and Theorem 2.3 are very similar to those of Lemma 2.1 and Theorem 2.1 and will not be illustrated here.

We demonstrate the use of this technique in the next section with the example from section 2.4.

2.6 *A NUMERICAL EXAMPLE TO ILLUSTRATE THE USE OF FILE PARTITIONING ON A DDB*

Using the same example in Section 2.4, we assume that both relations S and SP are partitionable into two segments each, with:

$P_S = P_{SP} = 2$

$l_{S_1} = l_{S_2} = 4500$

$l_{SP_1} = l_{SP_2} = 10000.$

We further assume that when a retrieval is made on a relation, all the segments of the relation must be accessed, that is, for $s,t \in \{S, SP\}$, $i \in \{1, 2, 3\}$ and $j \in \{1,2\}$,

$fr(s_j \,|\, q_{i,s}^{s}) = 1$

$fr(s_j \,|\, q_{i,S,SP}^{S,SP}) = 1$

$fr(s_j \,|\, u_{i,t}^{S,SP}) = 1$

We would like to see what is the effects of varying the fraction of updates that have to access multiple segments. For $i \in \{1,2,3\}$, let

$$fu^1 = fu(S_1 | u_{i,S}^{S}) = fu(S_2 | u_{i,S}^{S}) = fu(SP_1 | u_{i,SP}^{SP}) = fu(SP_2 | u_{i,SP}^{SP})$$

and

$$fu^2 = fu(S_1 | u_{i,S}^{S,SP}) = fu(S_2 | u_{i,S}^{S,SP}) = fu(SP_1 | u_{i,SP}^{S,SP}) = fu(SP_2 | u_{i,SP}^{S,SP})$$

That is, the fraction of updates that will access a particular segment of the relation is independent of the relation, but is dependent on the type of the updates, namely, single relation updates or multi-relation updates. The relation between fu^1 and fu^2 is shown in Figure 2.8. It is seen that the total operational costs after partitioning is always less than the cost without partitioning. However, due to the fact that there is a higher overhead in maintaining a larger number of files on the DDB, all the curves in Figure 2.8 will shift upward. Depending on the additional cost in the overhead, a threshold in fu^1 and fu^2 can be found, below which the scheme is cost-effective.

2.7 *COST REDUCTION ON THE PLACEMENT OF RELATIONS ON A DDB BY UTILIZING REDUNDANT INFORMATION AND FILE PARTITIONING*

The technique described in Sections 2.3 and 2.5 can be combined together to give a further reduction in the operational costs. Extra redundant information is first added to the relations in the DDB. These relations are then partitioned before they are allocated. Using the symbols defined before, we first discuss the case of two relations, a and b, which are partitioned in P_a and P_b segments. We assume that the multi-relation-reduced-single-relation queries behave in a similar fashion as the original multi-relation queries in accessing a segment of a relation, that is, the variables fr and fu defined for the multi-relation queries are identical for the variables fr and fu defined for the multi-relation-reduced-single-relation queries. Further, it is necessary to define for

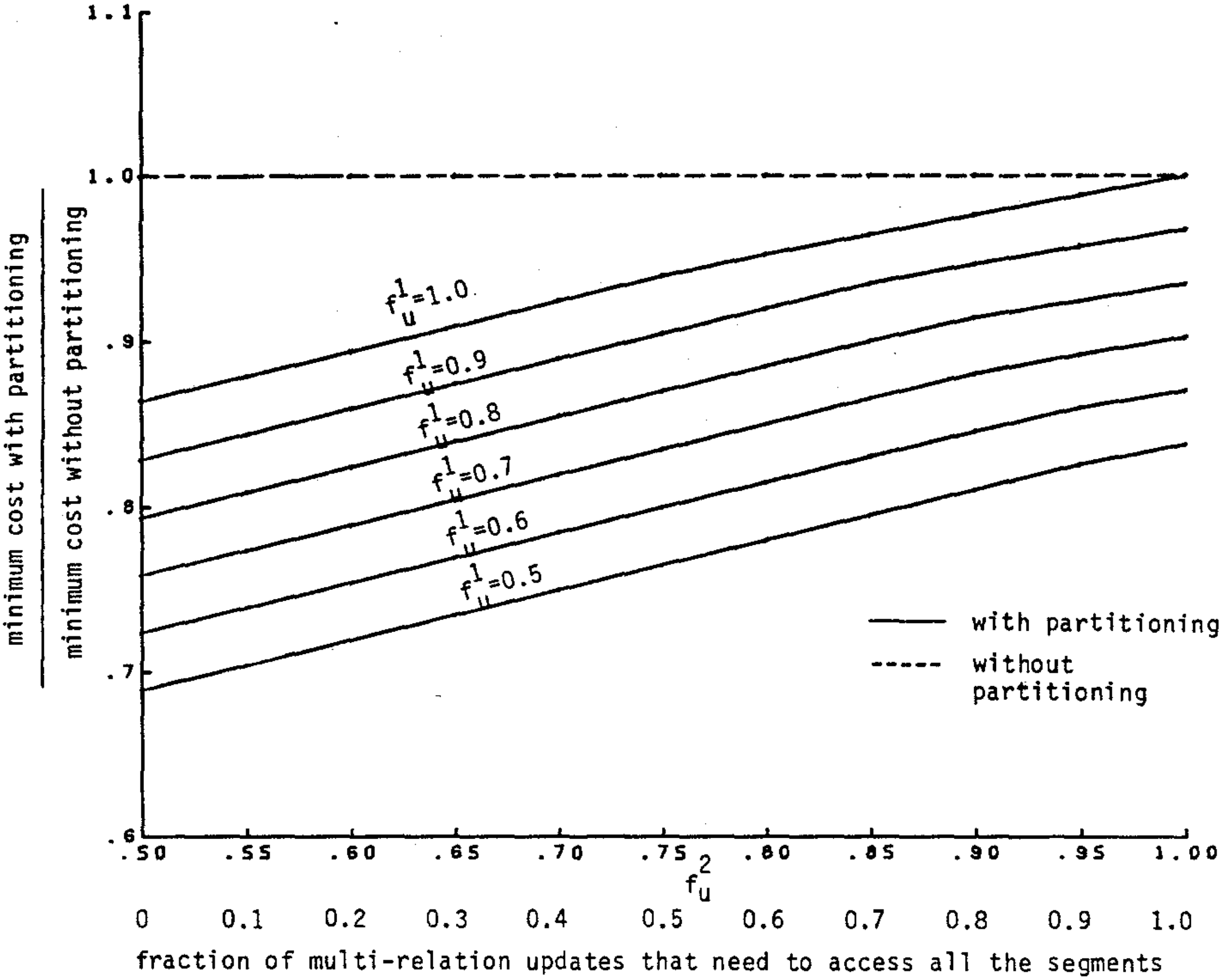


Figure 2.8 A Plot of Cost Ratio with respect to f_u^1 for various values of f_u^2 under File Partitioning

the updating of redundant information, the following symbols:

$fr^{rd}(t_j \mid u^{a}_{i,a})$ $[fr^{rd}(t_j \mid u^{b}_{i,b})]$ = fraction of updates on the redundant information that have to retrieve the j'th segment of relation t (t=a,b) in order to determine the redundant information to be updated, given that the single relation update rate is $u^{a}_{i,a}$ $(u^{b}_{i,b})$;

$fu^{rd}(t_j \mid u^{a}_{i,a})$ $[fu^{rd}(t_j \mid u^{b}_{i,b})]$ = fraction of updates on the redundant information that have to update the j'th segment of relation t (t=a,b) given that the single relation update rate is $u^{a}_{i,a}$ $(u^{b}_{i,b})$.

$fr^{rd}(t_j \mid u^{a,b}_{i,a})$ $[fr^{rd}(t_j \mid u^{a,b}_{i,b})]$ = fraction of updates on the redundant information that have to retrieve the j'th segment of relation t (t=a,b) in order to determine the redundant information to be updated given that the multi-relation update rate is $u^{a,b}_{i,a}$ $(u^{a,b}_{i,b})$.

$fu^{rd}(t_j \mid u^{a,b}_{i,a})$ $[fu^{rd}(t_j \mid u^{a,b}_{i,b})]$ = fraction of updates on the redundant information that have to update the j'th segment of relation t (t=a,b) given that the multi-relation update rate is $u^{a,b}_{i,a}$ $(u^{a,b}_{i,b})$.

l'_{t_j} = length of segment j (j=1,...,P_t) of relation t (t=a,b) after the redundant information has been added.

The optimization problem of placing the segments on the DDB is:

min (2.9)

$$\sum_{s=a,b}\sum_{k=1}^{P_s}\sum_{i=1}^{n}\sum_{j=1}^{n} q^{s}_{i,s} fr(s_k \mid q^{s}_{i,s})\alpha^{s}_{i,s} l'_{s_k} X^{s_k}_{i,j} S^{s}_{i,j}$$

$$+\sum_{s=a,b}\sum_{k=1}^{P_s}\sum_{i=1}^{n}\sum_{j=1}^{n} (1-\gamma^{a,b}_{i,a,b}) q^{a,b}_{i,a,b} fr(s_k \mid q^{a,b}_{i,a,b})\sigma^{a,b}_{i,s}\varepsilon^{a,b}_{i,s} l'_{s_k} X^{s_k}_{i,j} S^{s}_{i,j}$$

$$+\sum_{s=a,b}\sum_{k=1}^{P_s}\sum_{i=1}^{n}\sum_{j=1}^{n} \gamma^{a,b}_{i,a,b} q^{a,b}_{i,a,b} fr(s_k \mid q^{a,b}_{i,a,b})\alpha^{a,b}_{i,s} l'_{s_k} X^{s_k}_{i,j} S^{s}_{i,j}$$

$$+\sum_{s=a,b}\sum_{k=1}^{P_s}\sum_{i=1}^{n}\sum_{j=1}^{n} u^{s}_{i,s} fu(s_k \mid u^{s}_{i,s})\beta^{s}_{i,s} l'_{s_k} M^{s}_{i,j} Y^{s_k}_{j}$$

$$+ \sum_{s=a,b} \sum_{k=1}^{P_s} \sum_{i=1}^{n} \sum_{j=1}^{n} u_{i,s}^{a,b} \Bigg[\delta_{i,s}^{a,b} \sum_{t=a,b} \sum_{e=1}^{P_t} fr(t_e \mid u_{i,s}^{a,b}) \nu_{i,t}^{a,b} l_{t_e}^{'} X_{i,j}^{t_e} S_{i,j}^{t}$$

$$+ fu(s_k \mid u_{i,s}^{a,b}) \beta_{i,s}^{a,b} l_{s_k}^{'} M_{i,j}^{s} Y_j^{s_k} \Bigg]$$

$$+ \sum_{s=a,b} \sum_{i=1}^{n} \sum_{j=1}^{n} \eta_{i,a,b}^{s} u_{i,s}^{s} \Bigg[\sum_{t \neq s} \sum_{e=1}^{P_t} fr^{rd}(t_e \mid u_{i,s}^{s}) \alpha_{i,t}^{a,b} l_{t_e}^{'} X_{i,j}^{t_e} S_{i,j}^{t}$$

$$+ \sum_{t=a,b} \sum_{e=1}^{P_t} fu^{rd}(t_e \mid u_{i,s}^{s}) \xi_{i,t}^{t} l_{t_e}^{'} M_{i,j}^{t} Y_j^{t_e} \Bigg]$$

$$+ \sum_{s=a,b} \sum_{i=1}^{n} \sum_{j=1}^{n} \eta_{i,a,b}^{s} u_{i,s}^{a,b} \Bigg[\sum_{t \neq s} \sum_{e=1}^{P_t} fr^{rd}(t_e \mid u_{i,s}^{a,b}) \alpha_{i,t}^{a,b} l_{t_e}^{'} X_{i,j}^{t_e} S_{i,j}^{t}$$

$$+ \sum_{t=a,b} \sum_{l=1}^{P_t} fu^{rd}(t_e \mid u_{i,s}^{a,b}) \xi_{i,t}^{t} l_{t_e}^{'} M_{i,j}^{t} Y_j^{t_e} \Bigg]$$

$$+ \sum_{s=a,b} \sum_{k=1}^{P_s} \sum_{i=1}^{n} f_{i,s} l_{s_k}^{'} Y_i^{s_k}$$

subject to the constraints 2.7f to 2.7n with four additional constraints:

$$1 \leq \sum_{j=1}^{P_s} fr^{rd}(s_j \mid u_{i,t}^{t}) \leq P_s \qquad s,t \in \{a,b\}, \quad i \in \{1,\ldots,n\}$$

$$1 \leq \sum_{j=1}^{P_s} fu^{rd}(s_j \mid u_{i,t}^{t}) \leq P_s \qquad s,t \in \{a,b\}, \quad i \in \{1,\ldots,n\}$$

$$1 \leq \sum_{j=1}^{P_s} fr^{rd}(s_j \mid u_{i,t}^{a,b}) \leq P_s \qquad s,t \in \{a,b\}, \quad i \in \{1,\ldots,n\}$$

$$1 \leq \sum_{j=1}^{P_s} fu^{rd}(s_j \mid u_{i,t}^{a,b}) \leq P_s \qquad s,t \in \{a,b\} \quad i \in \{1,\ldots,n\}$$

The explanation of each term of Eq. 2.9 is similar to the corresponding term of Eq. 2.7.

A lemma and theorem similar to Lemma 2.1 and Theorem 2.1 can be proved for this problem.

LEMMA 2.4

Optimization problem 2.9 can be partitioned into P_a+P_b independent optimization sub-problems, one for each segment. The optimization sub-problem for segment s_k where $s \in \{a, b\}$, $k \in \{1, \ldots, P_s\}$ is

$$\min \sum_{i=1}^{n} Q_i^{s_k} \min_{j, Y_j^{s_k}=1} S_{i,j}^{s} + \sum_{i=1}^{n} \sum_{j=1}^{n} U_i^{s_k} M_{i,j}^{s} Y_j^{s_k} + \sum_{i=1}^{n} F_i^{s} Y_i^{s_k} \tag{2.10}$$

where

$$Q_i^{s_k} = \Big[q_{i,s}^{s} fr(s_k \mid q_{i,s}^{s}) \alpha_{i,s}^{s} + (1-\gamma_{i,a,b}^{a,b}) q_{i,a,b}^{a,b} fr(s_k \mid q_{i,a,b}^{a,b}) \sigma_{i,s}^{a,b} \varepsilon_{i,s}^{a,b}$$
$$+ \gamma_{i,a,b}^{a,b} q_{i,a,b}^{a,b} fr(s_k \mid q_{i,a,b}^{a,b}) \alpha_{i,s}^{a,b} + \sum_{t=a,b} u_{i,t}^{a,b} \delta_{i,t}^{a,b} fr(s_k \mid u_{i,t}^{a,b}) \nu_{i,s}^{a,b}$$
$$+ \eta_{i,a,b}^{\overline{s}} u_{i,\overline{s}}^{\overline{s}} fr^{rd}(s_k \mid u_{i,\overline{s}}^{\overline{s}}) \alpha_{i,s}^{a,b} + \eta_{i,a,b}^{\overline{s}} u_{i,\overline{s}}^{a,b} fr^{rd}(s_k \mid u_{i,\overline{s}}^{a,b}) \alpha_{i,s}^{a,b} \Big] l_{s_k}^{'}$$

$$U_i^{s_k} = \Big[u_{i,s}^{s} fu(s_k \mid u_{i,s}^{s}) \beta_{i,s}^{s} + u_{i,s}^{a,b} fu(s_k \mid u_{i,s}^{a,b}) \beta_{i,s}^{a,b}$$
$$+ \sum_{t=a,b} \eta_{i,a,b}^{t} u_{i,t}^{t} fu^{rd}(s_k \mid u_{i,t}^{t}) \xi_{i,s}^{s} + \sum_{t=a,b} \eta_{i,a,b}^{t} u_{i,t}^{a,b} fu^{rd}(s_k \mid u_{i,t}^{a,b}) \xi_{i,s}^{s} \Big] l_{s_k}^{'}$$

$$F_i^{s_k} = f_{i,s} l_{s_k}^{'}$$

$\overline{s} = t \in \{S, SP\}$ and $t \not= s$

subject to

$$\sum_{i=1}^{n} Y_i^{s_k} \geq 1$$

$$Y_i^{s_k} = 0, 1 \qquad i=1, \ldots, n$$

A generalization of Lemma 2.4 is to allow any number of relations in the DDB. This is shown in the following theorem.

THEOREM 2.4

The general problem of optimizing the placements of multiple relations on a DDB using additional redundant information and file partitioning can be decomposed into multiple sub-problems, one for the placement of each partition.

The proofs of Lemma 2.4 and Theorem 2.4 are very similar to those of Lemma 2.1 and Theorem 2.1 and will not be illustrated here.

The next section demonstrates the use of the combined technique with the same example discussed in Section 2.4.

2.8 *A NUMERICAL EXAMPLE TO ILLUSTRATE THE USE OF ADDITIONAL REDUN-*

DANT INFORMATION AND FILE PARTITIONING ON A DDB

Using the same values defined in Sections 2.4 and 2.6, we further assume for $s,t \in \{S,SP\}$, $i \in \{1,2,3\}$ and $j \in \{1,2\}$, that

$$fr^{rd}(t_j \,|\, u_{i,s}^{s}) = 1$$

$$fr^{rd}(t_j \,|\, u_{i,s}^{a,b}) = 1$$

$$fu^{1} = fu^{rd}(s_i \,|\, u_{i,s}^{s}) = fu^{rd}(s_i \,|\, u_{i,t}^{t})$$

$$fu^{2} = fu^{rd}(s_i \,|\, u_{i,s}^{S,SP}) = fu^{rd}(s_i \,|\, u_{i,t}^{S,SP})$$

The evaluations of the combined technique are shown in Figures 2.9 to 2.12. Comparing Figures 2.9 and 2.10 with Figures 2.5 and 2.6 under the assumption of $fu^1=fu^2=0.75$ (that is, 50% of the updates have to access the two segment together), it is seen that the combined technique gives a larger cost decrease than when redundancy is used alone. In fact, as seen from Figures 2.9 and 2.10, it is "almost" true that the combined technique is always more cost effective than the case when none of the techniques are used. On the other hand, the curves plotted in Figure 2.11 where $\delta=\eta=\gamma=0.5$, have a higher cost ratio than the curves plotted in Figure 2.8. This means that the use of the combined technique is worse than the case when partitioning is used alone. The explanation for this is because there is a large update cost for the additional redundant information and this is not offset by the cost decrease due to partitioning. However, if sufficient redundant information can be added to the system so that the retrieval cost can be further reduced, the total operational costs may drop. This is shown in Figure 2.12, where $\delta=\eta=\gamma$ have been reduced to 0.4. The curves in Figure 2.12 indicate a smaller cost than the curves in Figure 2.8.

We conclude that the combined technique is always better than the technique of using redundancy alone and is better than the partitioning technique only when δ, γ and η are "small enough".

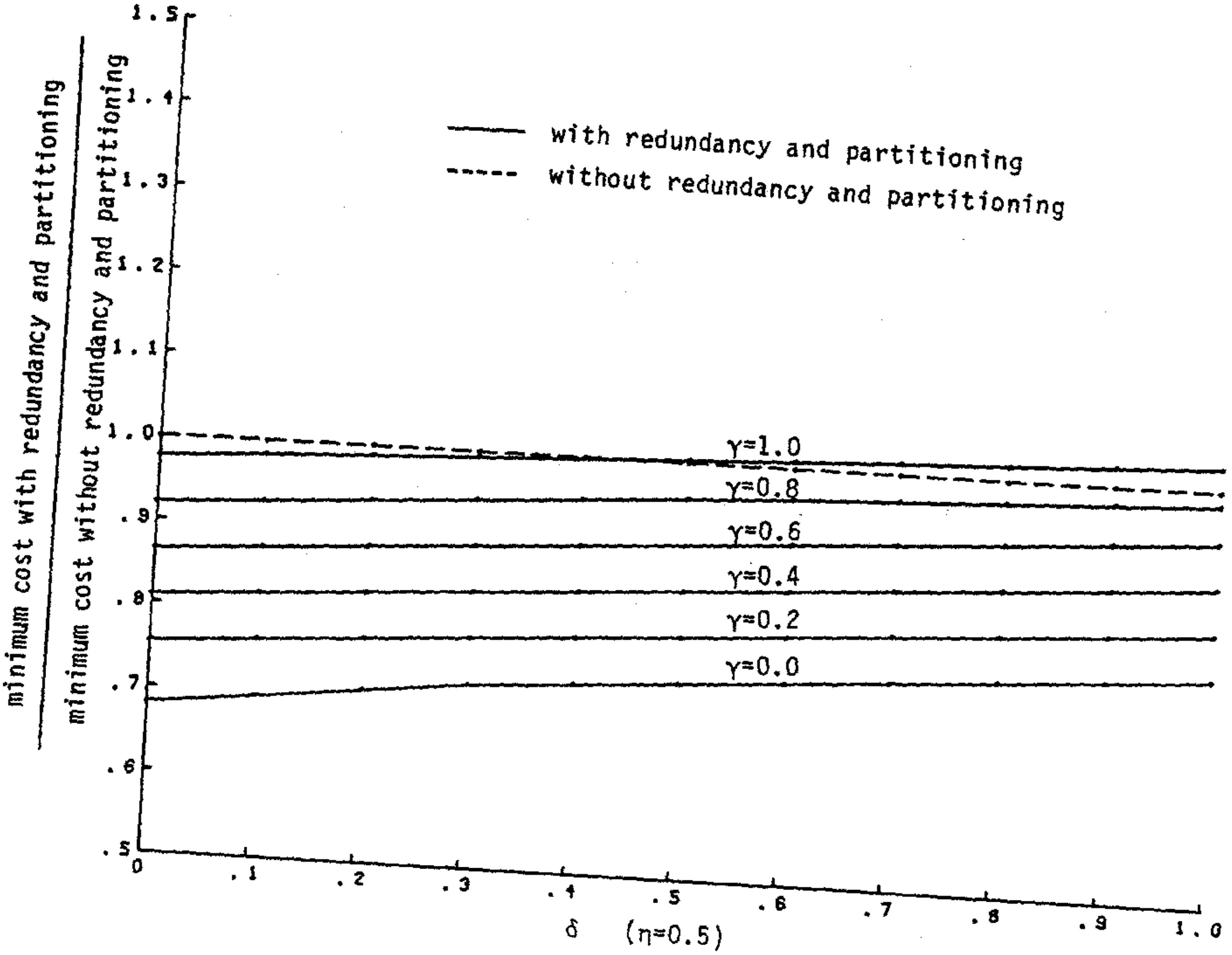


Figure 2.9 A Plot of Cost Ratio with respect to γ for various values of δ under the use of Redundancy and Partitioning ($f_u^1=f_u^2=0.75$)

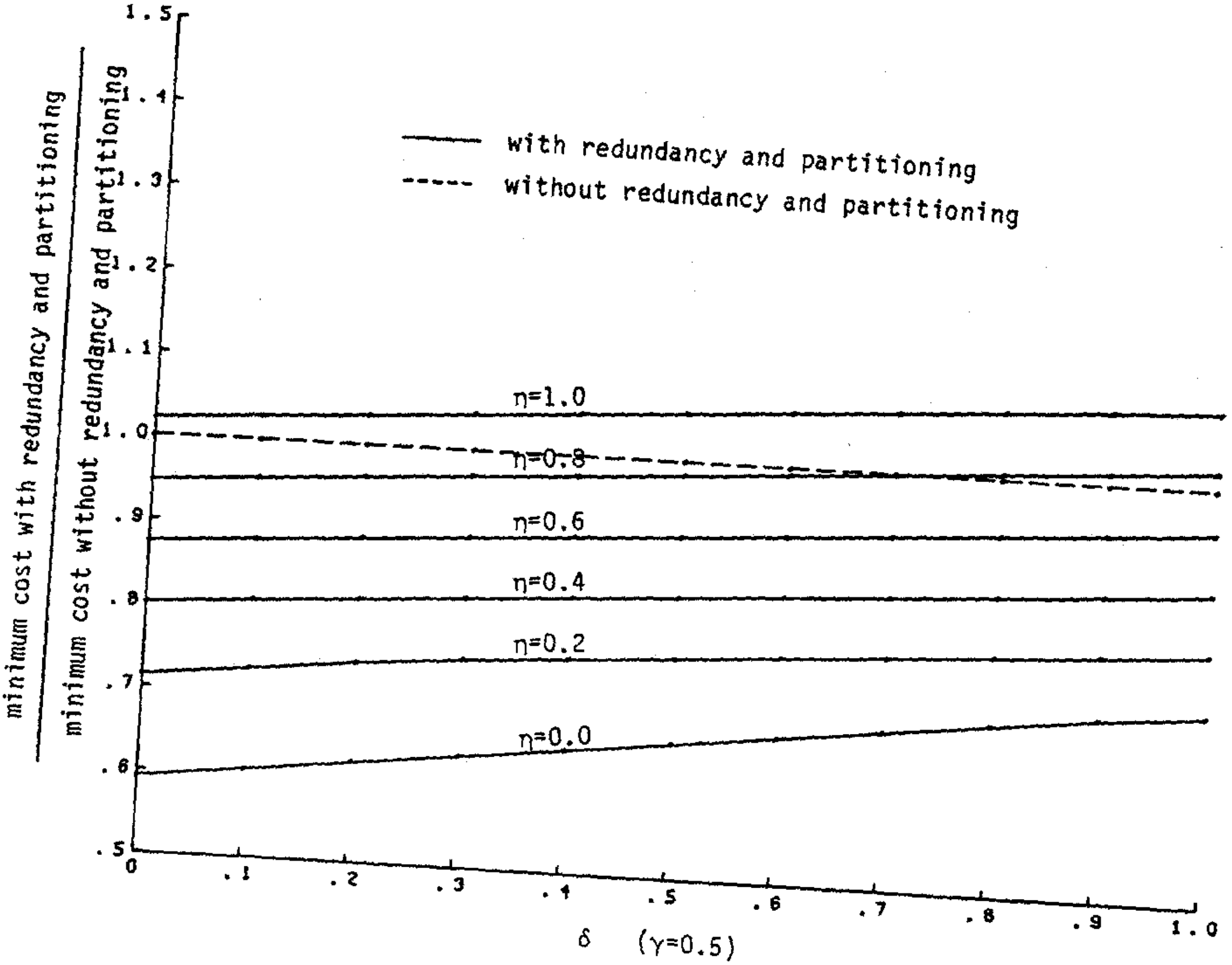


Figure 2.10 A Plot of Cost Ratio with respect to η for various values of δ under the use of Redundancy and Partitioning ($f_u^1=f_u^2=0.75$)

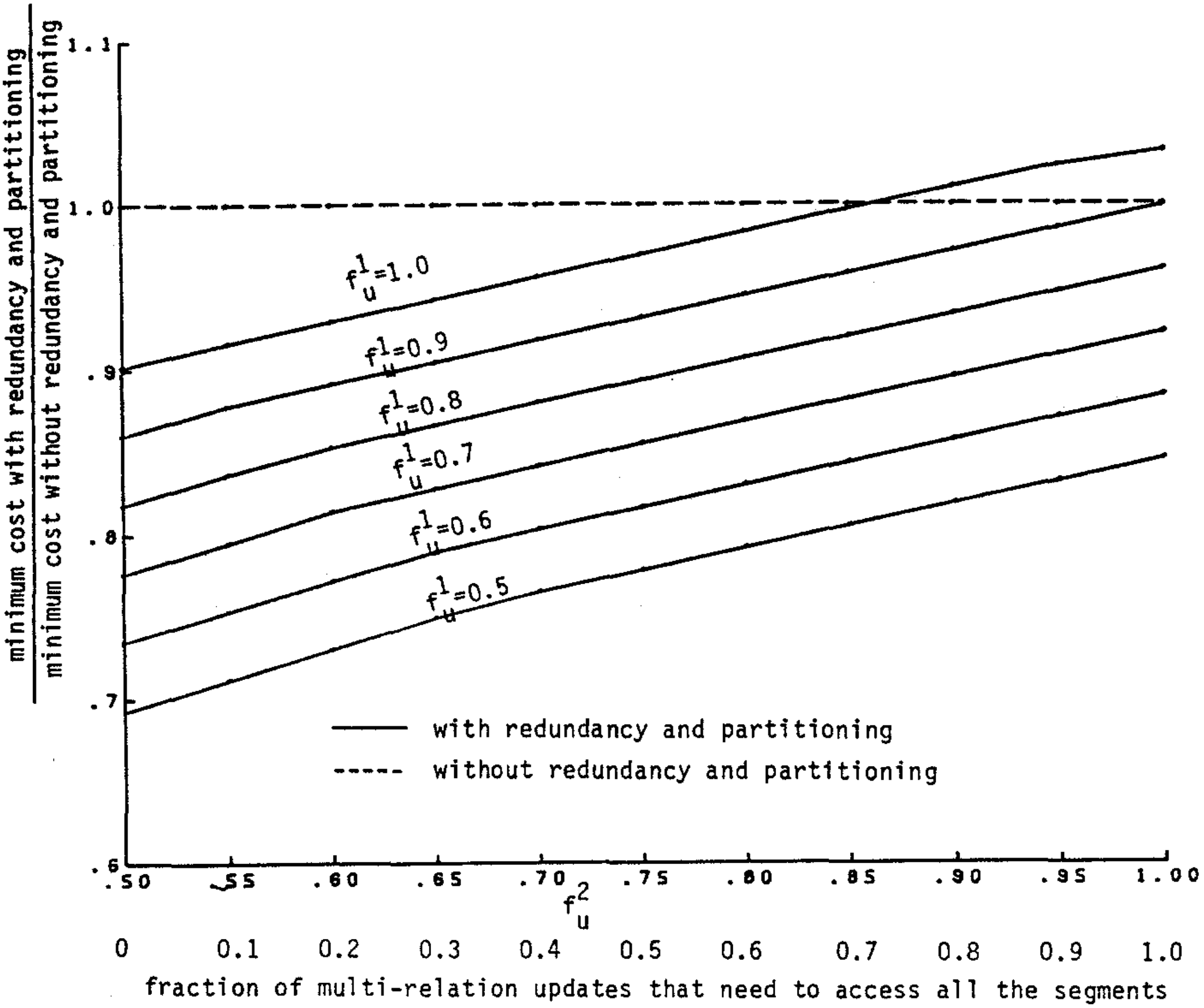


Figure 2.11 A Plot of Cost Ratio with respect to f_u^1 for various values of f_u^2 under the use of Redundancy and Partitioning ($\delta=\eta=\gamma=0.5$)

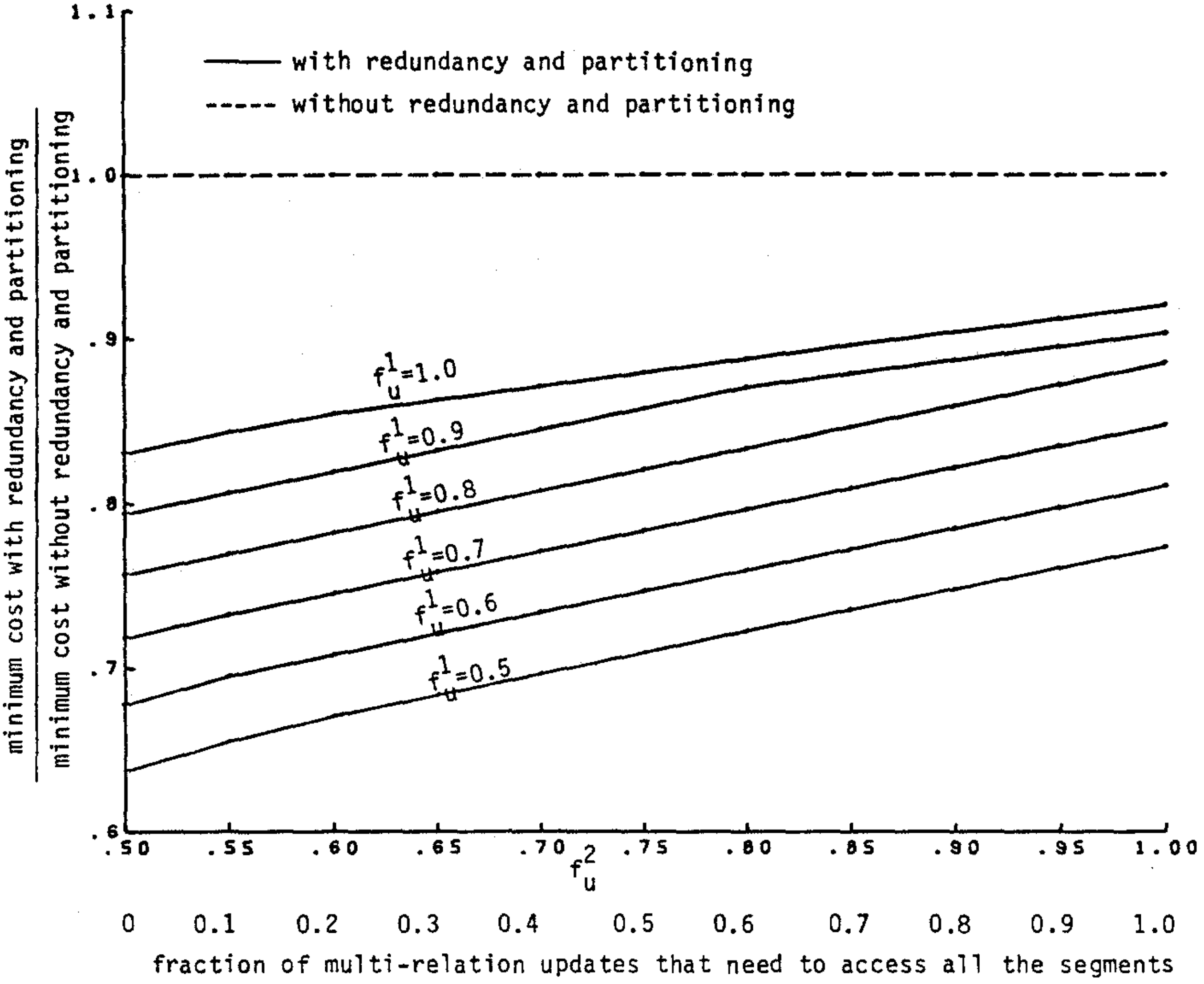


Figure 2.12 A Plot of Cost Ratio with respect to f_u^1 for various values of f_u^2 under the use of Redundancy and Partitioning (δ=η=γ=0.4)

2.9 *CONCLUSION*

In this chapter, we have studied the problem of optimal relation placements on a distributed relational data base. The objective of the problem is to minimize the total operational costs of the system and to allow query decomposition to be done more efficiently. The type of queries that can be made on a distributed relational data base are classified. It is seen that non-decomposable queries cause a lot of communication overhead on the system. Two techniques and a combination of these two techniques are analyzed in this chapter. By pre-analyzing the type of queries made on the DDB and the probability distribution of the data in the relations, the first technique introduces additional redundant information on the DDB so that non-decomposable queries can be made decomposable. The result is a decrease in the total retrieval cost and an increase in the total update cost. The second technique partitions the relations on the DDB into smaller segments which results in a decrease in the total update cost and an increase in the total retrieval cost. The third technique combines the above two techniques together. The total operational costs are going to drop if the total cost increase is offset by the total cost decrease. It is proven in this chapter that the problem of optimal relation (or segments of relation) placement on a DDB can be decomposed into multiple sub-problems, one for the placement of each relations (or segments). The result is a significant reduction in the complexity of the optimization problem. A simple example is used to illustrate some of the properties of these techniques. It must also be noted that a lot of generality is introduced in the development of these techniques and a lot of parameters are defined. However, most of these parameters are identical in general, and therefore, as illustrated in the examples, the number of parameters to be estimated on the system is relatively small.

After decomposing the placements of multiple relations into the placements of individual relations, it is necessary to study algorithms to perform the placements. This is the topic of discussion in Chapter 3.

3. *THE PLACEMENT AND MIGRATION OF MULTIPLE COPIES OF A FILE ON A DCS*

3.1 *INTRODUCTION*

In the last chapter, we have studied the placement of multiple relations on a DCS and have decomposed the problem into multiple sub-problems of placing multiple copies of each relation independently. In this chapter, we develop the theory and the techniques to place and to migrate multiple copies of a single file on the DCS. This is done by first showing that the file allocation problem and the dynamic file allocation problem (or file migration problem), which have been studied extensively in Computer Science, are isomorphic to two equally well known problems in operations research, called the single commodity warehouse location problem and the single commodity dynamic warehouse location problem. Due to this isomorphism, it is found that many techniques which have been developed for one problem can be applied to solve the other problem. Further, it is found that some techniques developed for one problem match very closely with techniques developed for the other problem. The implications of such a proof of isomorphism are further shown in sections 3.6 and 3.7. By combining some conditions devloped in both the file allocation problem and the warehouse location problem, we have developed a file placement heuristic which performs better than other heuristics proposed. The heuristic is tested on sample problems whose optimal solutions have been established previously in the literature. In studying the file migration problem, we have proved that it is NP-complete and have developed some conditions to indicate when file migration should be carried out.

3.2 *DEFINITION OF THE PROBLEM*

On a DCS, one of the important problems is to distribute or place the files so that they can be accessed efficiently. Chu has studied the optimal file allocation problem which is defined as follows: given a number of computers that process common information files, how can one allocate the files so that the allocation yields minimum overall operating costs [CHU69]. This problem is directed toward the optimal placement of multiple files on the DCS. Subsequently, many researchers have partitioned the problem of allocating multiple files to multiple problems of allocating individual files, e.g. [CAS72, LEV74, MOR77]. In Chapter 2 of this thesis, we have shown the decomposition of the optimal placements of multiple relations into multiple sub-problems, one for the optimal placement of each relation in a distributed relational data base. This single file allocation problem has been coined by Eswaran as the *File Allocation Problem (FAP)* [ESW74] which can be defined as: given a number of computers that process common information files, and users on the system that access these files, how can one allocate multiple copies of a file so that the allocation yields minimum overall operating costs. This is a very simple formulation in which all the constraints on the system are transformed into a common unit of cost which may include file access cost, multiple update cost, file storage cost and file migration cost. Different constraints may be reflected in the form of different costs. For example, a prohibitive route in the network is represented by a high acces cost. A more general problem is the *Dynamic File Allocation Problem (DFAP)* or the *File Migration Problem* in which the files are allowed to migrate over time in order to adapt to changing access requirements. It is assumed that the period for migration is fixed ahead of time and is not determined dynamically. There are other people who have studied variations of the general process and file allocation problem. Among them are Stone and Jenny, who have studied the allocations of processes on a multi-processor system [STO77a, STO78a, STO78b, JEN77, HOF78]; Loomis and Popek, who have introduced additional parameters such as the capability of a node on their model [LOO75, LOO76]; Mahmoud and Riordon, who have considered the file placements and the capacity assignments for links jointly [MAH76]. We concentrate in this chapter on the FAP defined by Eswaran (multiple copies of single file allocation problem) and the DFAP (an extension of FAP in which the placements vary over time).

3.3 *MOTIVATIONS FOR FILE PLACEMENT AND MIGRATION*

The major reason that multiple copies of a file are allocated to certain parts of the system at certain times and it is not necessary to keep a copy of every file at every node all the time is because users have localities of access. At any particular time, a file may be used by a group of users, and it will continue to be used by the same group for a certain length of time. For a particular user, the file that he wants to access may be available locally, in which case, he can access the file with very little cost. If the file is not available locally, he would have to pay a cost in terms of delay in accessing the file and also additional traffic in the network before he can make the access. It is under this situation that we should consider moving a copy of the file to his node. Introducing a new copy would also increase the cost in terms of storage space and the extra overhead in locking and concurrency control. Therefore, the decision of whether to introduce a new copy of a file involves a balance of the cost between the two cases. The costs, e.g. communication costs, storage costs, etc., are a function of the topology of the system, the storage sub-system at a node, the type of communication protocols used, and most importantly, the extensiveness of usage at a particular node. Some examples of the tariff for the usage of Telenet Data Communication Network are shown in Table 3.1 [TEL78]. For example, suppose the user uses a public dial-in service with local dial at 1200 bps, the cost that he has to pay (assuming 100% line utilization with 30% overhead) is $4.009 for 1 Kbyte of data. On the other hand, the storage costs on the system, with the

Table 3.1 Examples of Communication Costs on Telenet Data Communication Network (July 1, 1978) [*TEL78*]

Type of Service	Port Speed	Installation Charge ($)	Usage Charge ($)
Dedicate Access Facilities	50-300 bps	400	300/month
	1200 bps	500	340/month
	9600 bps	800	1100/month
Public Dial-in Service	Local Dial 110-300 bps	0	3.25/*hr*•
	Local Dial 1200 bps	0	3.25/*hr*•
	In-WATS 110-300 bps	0	15.00/hr
	In-WATS 1200 bps	0	15.00/hr
Private Dial-in Service	110-300 bps	320	160/*month*•
	1200 bps	340	215/*month*•
	TWX	300	210/*month*•
Private Dial-out Service	75-300 bps	420	300/*month*•
	TWX	420	300/*month*•

• Regular Service - $0.50 per thousand packets; each packet contains up to 128 characters of user data.

advances of low cost mass storage, are much smaller as compared with the communication costs. As an example, it costs $1.00/month to store 24 Kbytes of data on the disk of the CDC 6400 at the University of California, Berkeley. Therefore, the minimization of communication traffic on the DCS, in the expense of using additional storage by having multiple copies of the data, is a more important problem.

Before we show the proof of the isomorphism, we survey in the next two sections, some of the previous work on file allocation and warehouse location.

3.4 *PREVIOUS WORK ON THE FILE ALLOCATION PROBLEM*

Most of the previous work on file allocation is based on static distribution, that is, the allocation does not change with time. A typical method ,in dynamic distribution involves the application of a static algorithm whenever need arises. Levin has applied dynamic programming to migrate copies of a file over a multi-period horizon [LEV74]. He has also developed some conditions in order to reduce the number of solution vectors that have to be generated in each period. However, the static algorithms are usually very expensive to run in real time. Grapa and Belford remarked that a particular solution to this problem solved a thirty node problem in one hour on an IBM 360/91 computer [GRA77b]. The difficulty in optimization is also exemplified in [SIC77]. Moreover, the problem has been shown to be NP-complete [ESW74], i.e., a class of problems for which there is no known optimal algorithm with a computation time which increases polynomially with the size of the problem [KAR72]. The computation times for all known optimal algorithms for this class of problem increase exponentially with the problem size, i.e., if n represents the size of the problem, then the computation time goes up as k^n where k>1. In order to achieve a polynomial execution time, heuristics are generally used which sacrifice optimality for efficiency. A summary of the previous work in file allocation is shown in Table 3.2. Some of these studies introduce additional constraints on the model (e.g. link capacity, node capability). Basically, the algorithms for statically allocating multiple copies of a single file can be divided into two types: (1) mathematical programming and exhaustive searches and (2) heuristics.

(1) *Mathematical Programming and Exhaustive Searches*

This technique has been used by Chu [CHU69], Casey [CAS72], Levin and Morgan [LEV74, LEV75, MOR77], and Mahmoud and Riordon [MAH76]. Using the notations defined in Chapter 2, and is repeated here in Table 3.3, the formulation of the FAP is as follows[1]:

[1] Since we are considering a single file a, without ambiguity, all the subscripts and superscripts for a will be delected in the formulation.

Table 3.2 A Summary of the Previous work in File Placement/Migration

Mathematical Programming & Exhaustive Searches					Heuristic	
Chu [CHU69]	Casey [CAS72]	Levin & Morgan [LEV74, LEV75, MOR77]	Ghosh [GHO76]	Foster et. al. [FOS77]	Loomis & Popek [LOO75, LOO76]	Mahmoud & Riordon [MAH76]
Complete relations among objects; File access is poisson.	All objects independent	Only program-data relations exist between objects.	All objects independent.	Star network; All objects independent.	Complete Probabilistic relations among objects.	Independent objects; Query and return traffic divided equally among allocated nodes.
Storage cost; Transmission cost; File length; Request rate between files; Update rate between files; Maximum allowable access time; Storage capacity.	Storage cost; Query transmission cost; Update transmission cost; Query rate between nodes; Update rate between nodes.	Communication cost for query; Communication cost for update; Traffic rate for query/update from a node to a file via a program; Inter-period file migration cost.	Data base with multiple target segment types; Queries with multiple target segment types.	Queueing time & service time for transactions; Storage capacity; Average number of messages in network; Average local processing; Average file length; Access frequency; Hardware, software characteristics.	Inter-node transmission cost; Node capability; File length; Processing needs of file; Probability of a request accessing an object; Probability that a request/update is incident on a node; Probability of 2 objects processed in parallel.	Communication cost; File storage cost; Query/update traffic & corresponding return traffic for each file at each node; Availability requirements.
Integer Programming	Path search on cost graph	Path search on cost graph; dynamic programming	Combinatorial search through possible solutions	Queueing network algorithm; Integer programming	Clustering	Add-drop heuristic
Algorithm very complex; Consider delay from network queueing approach.	Algorithm efficient; Independence of objects reduces allocation of multiple files into single file.	Algorithm efficient; Definite access relations among objects reduce the allocation of multiple files to single file; Define conditions to reduce dynamic programming search	Maximize number of segments that query can retrieve in parallel from different nodes; Do not model communication delays.	Minimize difference from optimal branching probabilities; Algorithm complex.	Dynamic network behavior ignored; Maximize potential for parallelism.	Obtain both capacity assignment for links & file placements; Should consider query to be routed to nearest node & not distributed equally among all nodes.

Table 3.3 Mapping between the Defined Notations in this Thesis and Casey's Notations [CAS72]

Notations defined in this thesis for file a	Casey's notations	Explanation
I	I	= index set of nodes with a copy of the file;
n	n	= number of nodes in the DCS;
U_j	ψ_j	= update load originating at node j per unit time;
Q_j	λ_j	= query load originating at node j per unit time;
$S_{j,k}$	$d_{j,k}$	= cost of communication of one query unit from j to k;
$M_{j,k}$	$d_{j,k}$	= cost of communication of one update unit from j to k;
F_k	σ_k	= storage cost of file at k per unit time.

An optimal allocation for a given file is then defined as an index set I which minimizes the cost function.

$$C(I) = \sum_{j=1}^{n} \left[\sum_{k\in I} U_j M_{j,k} + Q_j \min_{k\in I} S_{j,k}\right] + \sum_{k\in I} F_k$$

By defining a control variable Y_j such that

$$Y_j = \begin{cases} 0 & j \not\in I \\ 1 & j \in I \end{cases}$$

The cost function can be written as:

$$C(I) = \sum_{j=1}^{n} \left[\sum_{k=1}^{n} U_j M_{j,k} Y_k + Q_j \min_{k\in I} S_{j,k}\right] + \sum_{k=1}^{n} F_k Y_k$$

The optimization problem for file placements is:

min

$$C(I) = \sum_{j=1}^{n} Q_j \min_{k\in I} S_{j,k} + \sum_{k=1}^{n} G_k Y_k \qquad (3.1)$$

subject to

$$Y_k = 0 \textit{ or } 1 \ (\textit{integer}) \qquad k=1,\dots,n$$

and

$$G_k = F_k + \sum_{j=1}^{n} U_j M_{j,k} \tag{3.2}$$

The quantity G_k has been introduced as Z_k in [GRA77b]. Optimization problem (3.1) can be solved by using integer programming techniques [GEO72]. Casey [CAS72] and Levin and Morgan [LEV74, MOR77] have used the hypercube technique to enumerate over a reduced set of possible solutions in order to find the optimum. However, the approach of using integer programming or exhaustive enumeration is only suitable when the problem size is small. Due to this difficulty, Grapa and Belford have done some pioneering work in developing three simple conditions to check whether a copy of a file should be placed at a node [GRA77b]. This reduces the complexity of the problem tremendously because many alternatives can be eliminated.

(2) *Heuristics*

Heuristics are "reasonable" search strategies which do not guarantee that the optimum solution can be found. Heuristics are usually interactive algorithms. A feasible solution can be generated. Users or some decision algorithm then has to decide whether to improve the solution or not and how to improve it. The decision algorithm is usually an add-drop algorithm in which perturbation is induced on the existing solution to see if a better solution can be obtained. Three of the most commonly used heuristics are (1) hierarchical designs; (2) clustering algorithms; and (3) add-drop algorithms.

(1) *Hierarchical designs*

This is a heuristic procedure in which attention is first restricted to the more important features of a system. In a file allocation problem, attention can first be restricted to geographical regions. After analysis has been performed and the files have been distributed to different geographical regions, attention can be directed to the less important details such as allocating files within a geographical region. This stepwise refinement procedure can continue down many levels. At each level of optimization, it is hoped that the effects on the optimization of the current level from the levels above and the levels below are very small. Nevertheless, iterations and design cycles may exist to refine the solution.

(2) *Clustering algorithms*

Clustering algorithms are horizontal design processes which have a similar objective as hierarchical algorithms, namely, to reduce the complexity of the analysis in a large system. In a DDB, clusters can be formed from geographical distribution of access frequencies. The files are then allocated to clusters. The file allocation within a cluster may further be refined as in hierarchical algorithms [LOO75, LOO76].

(3) *Add- drop algorithms*

In applying this algorithm, a feasible distribution of files is first found. The total cost of the system can be improved by successive addition or deletion of file copies. When a feasible solution with a lower cost is found, it is adopted as a new starting solution and the process continues. Eventually, a local optimum is reached in which addition or deletion does not reduce the cost. The whole procedure can be repeated with a different starting feasible solution and several local optima can be obtained. By taking the minimum of all the local minima obtained, it is hoped that we can get very close to the global optimum [MAH76].

The disadvantages of all these heuristics are that they usually find a local optimum instead of a global optimum and the validation is very difficult. The goodness of a heuristic is often measured by its computational complexity and by its average and worst case behavior. Because the average and the worst case are difficult to solve analytically, evaluations are generally done by simulations. Therefore it is possible that the heuristic performs satisfactorily for some example problems, but it may perform unpredictably for some other problems. Using the add-drop principle, a heuristic for the FAP is shown in Section 3.8.

3.5 *PREVIOUS WORK ON THE SINGLE COMMODITY WAREHOUSE LOCATION PROBLEM*

Although the development of DCS's is very recent, and the problem of file allocation in DCS's is rather new, a similar problem has been studied by many operations researchers a long time ago. As early as 1951, Dantzig used the simplex method to solve the transportation problem [DAN51]. In 1958, Baumol described a problem called the warehouse location problem [BAU58]. The problem was then studied by many people. There are several variations of the problem and all of them consider a single type of commodity on the system.

(1) *Simple plant location problem:*

Given a set of plants which can supply customers with goods and have no constraints on the amount shipped from any source, the problem is to determine the geographical pattern of plants' locations which will be most profitable to the company. The optimization is done by equating the marginal cost of warehouse operation with the transportation cost savings and incremental profits resulting from more rapid delivery. This problem has been studied in [MAN64, EFR66, SPI69, SNY71, ALC76]. Manne studied the use of "steepest ascent one point move algorithm" [MAN64]. Efroymson and Ray, Spielberg, Alcouffe and Muratet studied enumerative optimal algorithms [EFR66, SPI69, ALC76]. Snyder studied a special case of the plant location problem in which the paths connecting two plant locations lie on a rectangular grid [SNY71].

(2) *Single Commodity Warehouse location problem (SCWLP):*

Given a set of factories, a set of customers and a set of possible warehouse locations, the problem is to locate the warehouses so that the fixed and the operational costs of the system is minimum. A special form of the problem is to neglect the transportation costs from the factories to the warehouses and to consider only the transportation costs from the warehouses to the customers which then becomes the simple plant location problem. This problem has been studied in [KEU63, FEL66, KHU72]. Keuhn and Hamburger developed the add-drop heuristic for the problem [KEU63]. Feldman and Ray extended Keuhn and Hamburger's work to include non-linear fixed costs [FEL66]. Khumawala further extended Efroymson and Ray's work [EFR66] and applied branch and bound algorithm to solve the problem [KHU72].

(3) *Single Commodity Dynamic facility location problem (SCDWLP):*

This is a dynamic version of the simple plant location problem or the warehouse location problem, except that the locations of plants or warehouses are allowed to change over a planning horizon of r periods so as to adapt to changing demands of the customers. This problem, first proposed by Francis [FRA63], has been studied in [WES73, ERL74, SWE76, RAO77]. Wesolowsky and Erlenkotter studied the single facility migration problem [WES72, ERL74]. Sweenly and Tatham applied dynamic programming to solve the multi-facility migration problem [SWE76]. Rao and Rutenberg studied a dynamic multi-location problem in which time is continuous and demand can change at different rates [RAO77].

(4) *Capacitated warehouse location problem:*

Consider a set of warehouses with a finite and fixed capacity, the problem is to determine the warehouses' locations so that the customers' needs can be satisfied and the costs of the system is minimum. This problem has been studied in [SA 69, GIG73, AKI77]. Sa, Akinc and Khumawala solved the problem using branch and bound technique [SA 69, AKI77]. Giglio solved a special case of the SCDWLP in which capacity constraints are taken into account and demands are assumed to be growing at a decreasing rate.

(5) *Quadratic assignment problem:*

Given a set of plants in which certain fixed quantities of the single type of commodity are to be shipped between the plants, and a set of possible plant locations, the problem is to assign the plants to locations so that the total costs of the system is minimum. This problem appears in [KOO57, GIL62, ARM63, LAW63, HIL66b, GRA70, RIT72]. Armour and Buffa have presented a heuristic which considered pairwise exchanges of work centers and locations [ARM63]. Gilmore and Lawler have developed optimal algorithms which are computationally feasible for small problems [GIL62, LAW63]. Lawler's solution requires a large number of linear assignment problems to be solved. Hiller and Connors modified Gilmore and Lawler's algorithms and obtained a more efficient but sub-optimal algorithm [HIL66b]. Graves and Whinston solved the problem using a probabilistic branch and bound algorithm [GRA70].

Some of the problems defined above are more general than the others. In fact, problem (1) is a subset of problem (2) which in turn is a subset of problem (3). Problem (4) also contains problems (1) and (2). We are concerned in this paper with problems (1), (2) and (3). The formulations of problems (1) and (2) are identical. Using the notations of Efroymson and Khumawala [EFR66, KHU72], the SCWLP, with m potential warehouses (with unlimited capacity) and n customers, can be formulated as a mixed integer program as follows.

minimize

$$Z = \sum_{i,j} D_j t_{i,j} X_{i,j} + \sum_{i} F_i Y_i$$

subject to

$$\sum_{i \in N_j} X_{i,j} = 1 \qquad j=1,\ldots,n$$

$$0 \leq \sum_{j \in P_i} X_{i,j} \leq n_i Y_i \qquad i=1,\ldots,m$$

$$Y_i = 0 \; or \; 1 \;\; (integer) \qquad i=1,\dots,m$$

where

$t_{i,j}$ = the per unit cost which includes the FOB cost at the warehouse (i), the warehouse handling cost and the transportation cost from the warehouse to the customer (j);

D_j = the demand of customer j;

$X_{i,j}$ = the portion of D_j supplied from warehouse i;

F_i = the fixed cost associated with warehouse i;

N_j = set of warehouses which can supply customer j;

P_i = set of those customers that can be supplied by warehouse i;

n_i = number of elements in P_i;

$$Y_i = \begin{cases} 1 & \text{if } warehouse \; exists \; at \; site \; i \\ 0 & otherwise \end{cases} .$$

We assume that m=n and that every warehouse can supply every customer. Let:

I = index set of sites with a warehouse.

It has been shown in [ALC76] for j=1, ..., n that:

$$X_{i,j} = \begin{cases} 1 & \text{if } t_{i,j} = \min_{k \in I} t_{k,j}, \; i \in I \\ 0 & otherwise \end{cases}$$

That is, the commodity will be shipped to a customer from a warehouse with the minimum transportation costs. The optimization problem can be rewritten as:

minimize

$$Z = \sum_{j=1}^{n} D_j \min_{k \in I} t_{k,j} + \sum_{i=1}^{n} F_i Y_i \tag{3.3}$$

subject to

$$Y_i = 0, \; 1 \;\; (integer) \qquad i=1,\dots,n$$

In solving the warehouse location problem, many techniques have been developed. Substantial evaluation results can be found on some example warehouse location problems in the literature.

3.6 *THE ISOMORPHISM BETWEEN FILE ALLOCATION AND SINGLE COMMODITY WAREHOUSE LOCATION*

After defining the (D)FAP and the SC(D)WLP, we are ready to prove the following theorem.

THEOREM 3.1

The FAP and the SCWLP are isomorphic and the DFAP and the SCDWLP are isomorphic.

Proof

The theorem can be proved by associating the variables of the FAP with the variables of the SCWLP and similarly, the variables of the DFAP with the variables of the SCDWLP. This association is shown in Table 3.4. An alternative way to prove the theorem is to notice that Equations 3.1 and 3.3 are actually identical with only a change of variables. The mapping of the variables are also shown in Table 3.4.

Q.E.D.

Using the isomorphism result, we have shown the equivalence of these two problems. Therefore all the results available to operations researchers are available to computer scientists and vice versa. The implications are further illustrated in the next section.

3.7 *IMPLICATIONS OF THE ISOMORPHISM BETWEEN THE (D)FAP AND THE SC(D)WLP*

Because the FAP and the SCWLP have been studied in different directions for a long time, techniques developed for one problem can be used to solve the other problem. The techniques developed for the general warehouse location problem can be used to solve the FAP and the DFAP. These include the add-drop

Table 3.4 Mapping between the (D)FAP and the SC(D)WLP

FAP		SCWLP	
Locations of computers	n	Possible warehouse sites	n
Locations of file	I	Locations of warehouse	I
Access for a file		Commodity flow	
Amount of access at j	Q_j	Customer demand at j	D_j
Per unit cost of communicating one query unit from j to k	$S_{j,k}$	Per unit cost of shipping commodity from plant to warehouse j and from warehouse j to customer k	$t_{j,k}$
File storage cost + multiple update cost for file at node k	C_k	Fixed cost of opening a warehouse at site k	F_k
File migration		Warehouse relocation	
Cost of migrating a copy of the file from j to k		Cost of relocating a warehouse from site j to site k	

technique developed in [KEU63, ARM63, FEL66] which is a heuristic of complexity $O(n^4)$ and generates sub-optimal solutions; the branch and bound algorithms used in [EFR66, SA 69, KHU72, AKI77] which exhaustively enumerate over a reduced set of possible solutions in order to obtain the optimal allocations and the running time depends on the bounding and the branching criteria used; the probabilistic branch and bound algorithm used in [GRA70] which is similar to the branch and bound technique but it uses probabilistic estimation to generate a lower bound; the direct search or implicit enumeration algorithm used in [SPI69, ALC76]; the steepest ascent algorithm used in [MAN64} which is a sub-optimal steepest ascent one point move algorithm; the dynamic programming method used in [SWE76] in which some conditions are developed to reduce the number of solution vectors searched; the heuristic developed in [HIL66b]; and the polynomial algorithms for some special cases, e.g. a plant location problem on a grid-like network is solved in [SNY71], a one facility plant migration problem is solved in [WES73]. Similarly, there are techniques developed in the FAP and the DFAP which can be used to solve the general warehouse location problem. These include the hypercube technique developed in [CAS72, LEV74, MOR77] which is essentially the same as Alcouffe and Muratet's optimal algorithm [ALC76] and is an implicit enumeration with conditions to discontinue unnecessary searches; the clustering technique used in [LOO75]; the dynamic programming method used in [LEV74] to solve for the optimal migration sequence of copies of a file; and the max-flow-min-cut network flow technique developed in [STO77a, STO78a, STO78b], which can be used to solve a special case of the SCQAP[2].

Besides the fact that techniques developed for both problems are interchangeable, there are instances where techniques developed for one problem match very closely with techniques developed for the other problem. These are stated in the following three corollaries.

COROLLARY 3.1

Two of the three conditions derived by Grapa and Belford [GRA77b] for a file to be placed or not to be placed at a node are weaker than the conditions derived by Efroymson and Ray [EFR66] for a warehouse to be opened or closed.

Proof

Before the conditions can be stated, some additional symbols must be defined. Let:

$K_0 = \{j : Y_j = 0\}$;

$K_1 = \{j : Y_j = 1\}$;

$K_2 = \{j : Y_j = \mathit{unassigned}\}$.

In the FAP, K_1, K_0 represent the set of nodes with and without a copy of the file, and K_2 represents the remaining nodes in the system. In the SCWLP, K_1, K_0

[2] The proof of this is shown in Appendix A

represent the set of sites for which a warehouse is opened and closed and K_2 represents the set of the remaining sites.

Two of the three Grapa and Belford's conditions for a file to be placed or not to be placed at a node are[3] [GRA77b]:

For i$\in K_2$:

$$Y_i = 1 \text{ if } Q_i \min_{\substack{j \in K_1 \cup K_2 \\ j \neq i}} (S_{i,j} - S_{i,i})_+ > G_i \tag{3.4}$$

$$Y_i = 0 \text{ if } \sum_{j=1}^{n} Q_j \max_{k \in K_1 \cup K_2} (S_{j,k} - S_{j,i}) < G_i \tag{3.5}$$

where:

$$(f)_+ = \begin{cases} f & \text{if } f \geq 0 \\ 0 & \text{if } f < 0 \end{cases}$$

Two of the three Efroymson and Ray's conditions for a warehouse to be opened or closed are[4] [EFR66]:

$$Y_i = 1 \text{ if } \sum_{j=1}^{n} Q_j \min_{\substack{k \in K_1 \cup K_2 \\ k \neq i}} (S_{j,k} - S_{j,i})_+ > G_i \tag{3.6}$$

$$Y_i = 0 \text{ if } \sum_{j=1}^{n} Q_j \min_{k \in K_1} (S_{j,k} - S_{j,i})_+ < G_i \tag{3.7}$$

In order to show that condition (3.4) is weaker than condition (3.6) and condition (3.5) is weaker than condition (3.7), it is necessary to show:

(a) $\sum_{j=1}^{n} Q_j \min_{\substack{k \in K_1 \cup K_2 \\ k \neq i}} (S_{j,k} - S_{j,i})_+ \geqq Q_i \min_{\substack{j \in K_1 \cup K_2 \\ j \neq i}} (S_{i,j} - S_{i,i})_+$

(b) $\sum_{j=1}^{n} Q_j \min_{k \in K_1} (S_{j,k} - S_{j,i})_+ \leqq \sum_{j=1}^{n} Q_j \max_{k \in K_1 \cup K_2} (S_{j,k} - S_{j,i})$

To prove

(a) L.H.S. $= Q_i \min_{\substack{k \in K_1 \cup K_2 \\ k \neq i}} (S_{i,k} - S_{i,i})_+ + \sum_{\substack{j=1 \\ j \neq i}}^{n} Q_j \min_{\substack{k \in K_1 \cup K_2 \\ k \neq i}} (S_{j,k} - S_{j,i})_+$

[3] Equation (3.4) has been augmented by the term $S_{i,i}$ on the R.H.S. because the original condition of Grapa and Belford is not correct when $S_{i,i} > 0$.

[4] The variables in the following two conditions have been transformed into the corresponding variables in the FAP with the use of Table 3.3. In the original Efroymson and Ray's conditions, $j \in P_i$ which is the set of customers that can be supplied from plant (or warehouse) i. We have made the assumption that all the customers can be supplied from any plant and therefore $j \in \{1,\dots,n\}$. Note that $t_{j,k}$ in the SCWLP corresponds to $S_{k,j}$ in the FAP.

$$\geq Q_i \min_{\substack{j \in K_1 \cup K_2 \\ j \neq i}} (S_{i,j} - S_{i,i})_+$$

$$= \text{R.H.S.}$$

(b) Comparing term by term, we would like to show that

$$\min_{k \in K_1} (S_{j,k} - S_{j,i})_+ \leq \max_{k \in K_1 \cup K_2} (S_{j,k} - S_{j,i})$$

There are two possibilities (note that $i \in K_2$):

If $\min_{k \in K_1} S_{j,k} \geq S_{j,i}$ then $\min_{k \in K_1} (S_{j,k} - S_{j,i})_+ = \min_{k \in K_1} (S_{j,k} - S_{j,i})$

$$\leq \max_{k \in K_1 \cup K_2} (S_{j,k} - S_{j,i})$$

If $\min_{k \in K_1} S_{j,k} < S_{j,i}$ then $\min_{k \in K_1} (S_{j,k} - S_{j,i})_+ = 0$

$$\leq \max_{k \in K_1 \cup K_2} (S_{j,k} - S_{j,i})$$

Therefore

$$\sum_{j=1}^{n} Q_j \min_{k \in K_1} (S_{j,k} - S_{j,i})_+ \leq \sum_{j=1}^{n} Q_j \max_{k \in K_1 \cup K_2} (S_{j,k} - S_{j,i})$$

This proves that the two Grapa and Belford's conditions are weaker than the corresponding Efroymson and Ray's conditions. It must be noted that the third condition derived by Grapa and Belford has no corresponding counterparts in the SCWLP and therefore may be useful in the SCWLP. The summary of these conditions are shown in Table 3.5.

Q.E.D.

By using the stronger Efroymson and Ray's conditions, a larger set of nodes can be pre-assigned to have or not to have a copy of the file than by using Grapa and Belford's conditions. This may save a lot of computation time in enumerating some possible assignments which cannot be pre-assigned using Grapa and Belford's conditions.

COROLLARY 3.2

The dynamic programming method for file migration used by Levin [LEV74] is

Table 3.5 Summary of Conditions for Placement and Non- placement of a file at node $i \in K_2$
(The first three conditions are from [EFR68]; the last condition is from [GRA77b].)

Condition	Rule
a	$Y_i = 1$ if $\sum_{j=1}^{n} Q_j \min_{\substack{k \in K_1 \cup K_2 \\ k \neq i}} (S_{j,k} - S_{j,i})_+ > G_i$
b	$Y_i = 0$ if $\sum_{j=1}^{n} Q_j \min_{k \in K_1} (S_{j,k} - S_{j,i})_+ < G_i$
c	If $\min_{k \in K_1} (S_{j,k} - S_{j,i}) < 0 \quad j \in \{1, \ldots, n\}$, then n_i is reduced by 1
d	$Y_i = 0$ if $G_i - G_k > \sum_{j=1}^{n} Q_j (S_{j,k} - S_{j,i})_+$

similar to the dynamic programming method for dynamic warehouse location used by Sweenly and Tatham [SWE76].

Proof

In [LEV74], Levin has developed a method of dynamically migrating copies of a file over a multi-period horizon. The technique uses the basic dynamic programming procedure, but additional conditions on costs are defined in order to reduce the number of solution vectors that have to be generated in each period. The conditions are defined so that the reduced set of solution vectors always include the optimum. On the other hand, Sweenly and Tatham also have used dynamic programming to solve the multi-period warehouse location problem. In order to reduce the number of solution vectors that have to be generated in each period, an upper bound is determined first. All solution vectors with values less than the upper bound are generated and ranked for each period. Dynamic programming is applied to find a new upper bound v^*. If $v^=$ is the sum of optimum solutions for each period without the relocation costs and $K=v^*-v^\infty$, then it is proven that additional solution vectors have to be generated for periods where the difference between the best and the worst solutions is less than K. A solution vector in a period is obtained by solving an integer program. It is difficult to determine the number of solutions to be generated in each period. However, some fixed number may be selected ahead of time based on the previous knowledge obtained. Although both techniques do not give any performance results on the number of solutions that have to be generated in each period, it seems that Levin's solution is easier to apply because it is not necessary to solve an integer program in order to obtain a solution. However, a smaller number of solutions may be generated using Sweenly and Tatham's technique, but it may be necessary to go through several iterations before the optimum solution is contained in the solution vectors, whereas using Levin's technique, the reduced set of solutions vectors always contain the optimum. The solutions that these two techniques give may not be identical and the practical benefits between these two methods can only be distinguished when they are applied on realistic problems. More evaluations are necessary before any quantitative judgement can be made between the two techniques.

Q.E.D.

COROLLARY 3.3

The hypercube technique developed by Casey [CAS72] and Levin and Morgan [LEV74, MOR77] and the condition used to discontinue the search, are identical to the algorithm and condition developed by Alcouffe and Muratet [ALC76].

Proof

The hypercube technique was first introduced by Casey [CAS72] (a later version was developed by Levin and Morgan [LEV74, MOR77]) to enumerate over all the possible combinations of allocations in order to find the optimal allocations.

A condition is developed to discontinue the search whenever the objective function [CAS72] (the sum of the query and the storage costs [LEV74, MOR77]) does not decrease after a file copy is added to an arbitary assignment at a node. A similar condition is also developed by Alcouffe and Muratet [ALC76]. However, the algorithm used by Alcouffe and Muratet is slightly different. They started their search from an assignment in which every warehouse is opened. This corresponds to the case in which every node has a copy of the file. In Casey's or Levin's algorithm, the search is started with the assignment in which every node does not have a copy of the file. However, the basic underlying principle of these two algorithms are still identical.

Q.E.D.

In conclusion, as a result of the proof of isomorphism, we have found that many techniques developed for both problems are inter-changeable and that some techniques developed for one problem match very closely with techniques developed for the other problem. It is therefore possible to study these two problems in an integrated fashion in the future. In the next section, we will use the conditions in Table 3.5 to develop a heuristic for the FAP.

3.8 *A HEURISTIC FOR THE FAP - Algorithm 3.1*

In this section, we propose a heuristic to solve the FAP. The search for an optimal solution is sometimes too time-consuming or impossible. Many of the optimal search techniques in the SCWLP are of branch and bound type and they are applicable to problems of moderate size. One way to reduce the execution time of a branch and bound algorithm is to develop some criteria so that many of the branches in the branch and bound tree can be systematically eliminated although the result obtained may not be optimal. In the heuristic we are going to discuss, several alternative criteria have been investigated. Essentially, the heuristic is a greedy algorithm which starts with all the nodes unassigned. It first applies the conditions of Table 3.5 to see if any node can be assigned without any enumeration. After all these nodes have been assigned, it comes to a point at which it has to decide what node to extend the assignment and whether or not to assign a copy of the file there. It does this by extending the current assignment by one node. For each of these extended assignments, there are two possibilities, either to assign or not to assign a copy of the file there. Therefore, there are altogether $2*|K_2|$ possible assignments which results in $2*|K_2|$ candidate problems. (The state of a candidate problem is made up of the states of allocation of the n different nodes on the DCS. In general, the n nodes of the DCS can be partitioned into three sets, K_0, K_1 and K_2.) For each of the candidate problems, a representative value is calculated. The function of the representative value is to estimate the minimum of the candidate problem without actually enumerating over all the allocations for the unassigned nodes. Based on these $2*|K_2|$ representative values, the selection criterion selects the node and decide whether or not to assign a copy of the file there. After this assignment has been made, the algorithm comes to a point at which it is ready to check for the conditions of Table 3.5 again and therefore it repeats the steps described above until all the nodes have been assigned. The general steps of the algorithm are shown in Figure 3.1. We discuss each of these steps briefly here.

M-1 This is to initialize the candidate problem - all nodes are unassigned at this point. The candidate list, which is a list of states, and is made up of the sets K_0, K_1, K_2 and its corresponding representative value, is assigned the empty set.

M-2-5 These four steps essentially achieve the following: a node is selected from the un-assigned set, K_2, and is assigned a copy or not assigned a copy of the file. A representative value is calculated for each of the candidate problems. The computed representative value and the corresponding assignments are attached to the candidate list. These

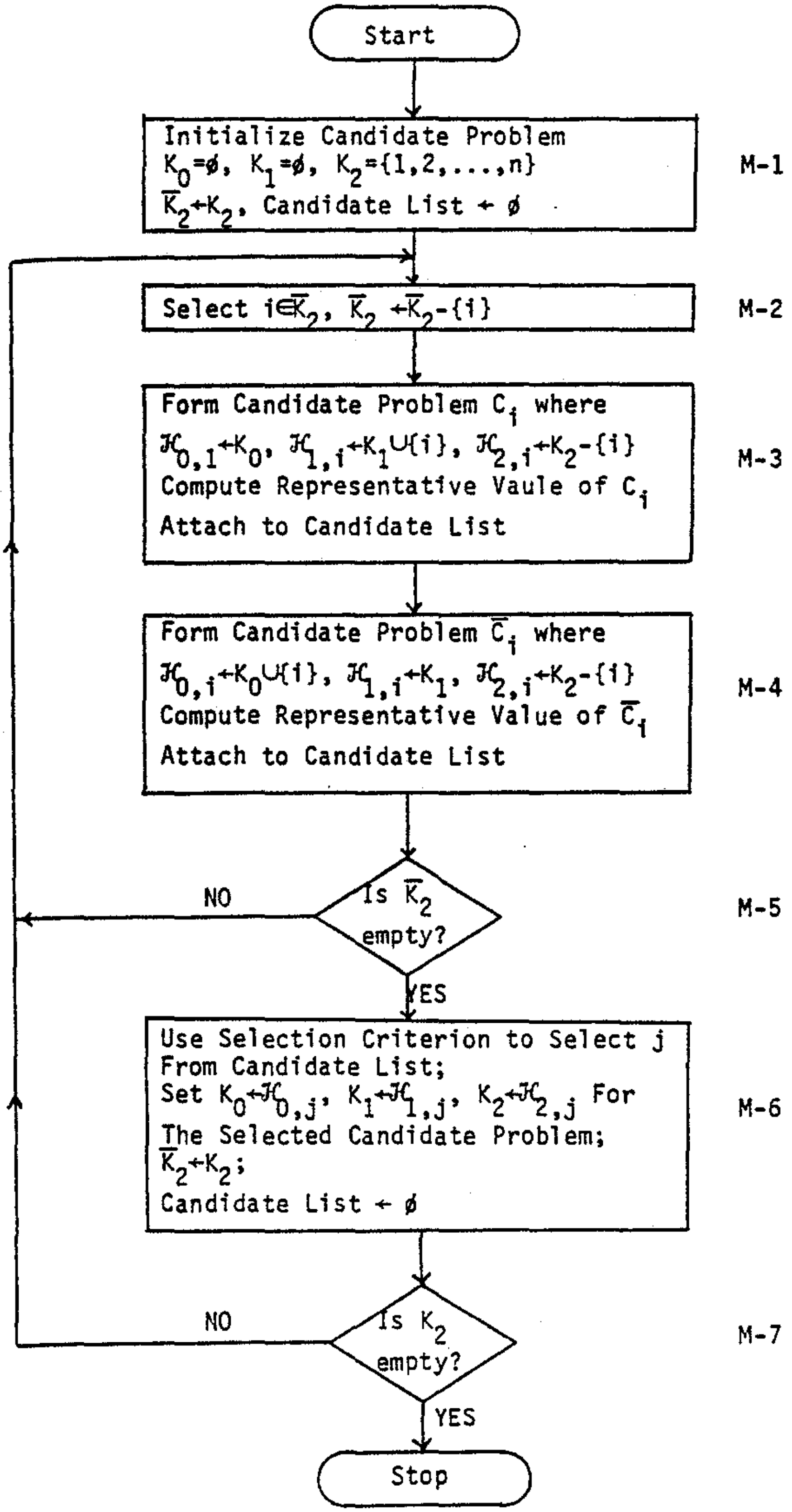

Start
Initialize Candidate Problem
$K_0=\phi$, $K_1=\phi$, $K_2=\{1,2,\ldots,n\}$
$\overline{K}_2 \leftarrow K_2$, Candidate List ← ϕ
M-1
Select $i \in \overline{K}_2$, $\overline{K}_2 \leftarrow \overline{K}_2-\{i\}$
M-2
Form Candidate Problem C_i where
$\mathcal{K}_{0,i} \leftarrow K_0$, $\mathcal{K}_{1,i} \leftarrow K_1 \cup \{i\}$, $\mathcal{K}_{2,i} \leftarrow K_2-\{i\}$
Compute Representative Vaule of C_i
Attach to Candidate List
M-3
Form Candidate Problem $\overline{C}_i$ where
$\mathcal{K}_{0,i} \leftarrow K_0 \cup \{i\}$, $\mathcal{K}_{1,i} \leftarrow K_1$, $\mathcal{K}_{2,i} \leftarrow K_2-\{i\}$
Compute Representative Value of $\overline{C}_i$
Attach to Candidate List
M-4
NO
Is $\overline{K}_2$ empty?
M-5
YES
Use Selection Criterion to Select j From Candidate List;
Set $K_0 \leftarrow \mathcal{K}_{0,j}$, $K_1 \leftarrow \mathcal{K}_{1,j}$, $K_2 \leftarrow \mathcal{K}_{2,j}$ For The Selected Candidate Problem;
$\overline{K}_2 \leftarrow K_2$;
Candidate List ← ϕ
M-6
NO
Is K_2 empty?
M-7
YES
Stop

Figure 3.1 File Assignment Algorithm

steps are then repeated for each node in K_2.

M-6 This step selects, from the candidate list, the candidate problem and the corresponding assignment of nodes using the selection criterion, and uses it for the next iteration. Steps M-2 to M-6 therefore have selected a node and have decided whether a copy should be placed at that node. This node is removed from the K_2 list.

M-7 The steps M-2 to M-6 are repeated until the K_2 list is empty.

There are two basic parts of the algorithm, the selection criterion and the computation of the representative value, and they are discussed here.

S1 *The selection criterion;*

S1a Select from the candidate list, the candidate problem with the minimum representative value;

S1b Select from the candidate list, the two candidate problems for which node i is extended, that have the maximum difference between the representative values of $Y_i=0$ and $Y_i=1$. From these two candidate problems, select the candidate problem with the minimum representative value.

R1 *The computation of the representative value;*

R1a A lower bound is computed by solving the linear program (Eq. 3.1) without the integrality constraints. (This has been derived earlier by Efroymson and Ray [EFR66]. See Appendix B for the derivation.);

R1b The expected value of the candidate problem is computed by assuming that each of the remaining un-assigned nodes has equal probability of having or not having a copy of the file (see Appendix C for the derivation);

Using the two selection criteria and the two types of representative values, there are four different versions of the algorithm:

1. MINLB - minimum lower bound (S1a, R1a);
2. MINE - minimum expected value (S1a, R1b);
3. MAXDLB - minimum lower bound for a node i with the maximum difference in lower bounds between $Y_i=0$ and $Y_i=1$ $(i \in K_2)$ (S1b, R1a);
4. MAXDE - minimum expected value for a node i with the maximum difference in expected values between $Y_i=0$ and $Y_i=1$ $(i \in K_2)$ (S1b, R1b);

To further illustrate the steps of the algorithm, it is applied on Casey's 5 node example [CAS72].

Suppose the following matrix represents the query cost $S_{i,j}$ for a five-node system.

$$S = \begin{bmatrix} 0 & 6 & 12 & 9 & 6 \\ 6 & 0 & 6 & 12 & 9 \\ 12 & 6 & 0 & 6 & 12 \\ 9 & 12 & 6 & 0 & 6 \\ 6 & 9 & 12 & 6 & 0 \end{bmatrix}$$

Let

$$Q = [Q_i] = [\ 24\ 24\ 24\ 24\ 24]$$

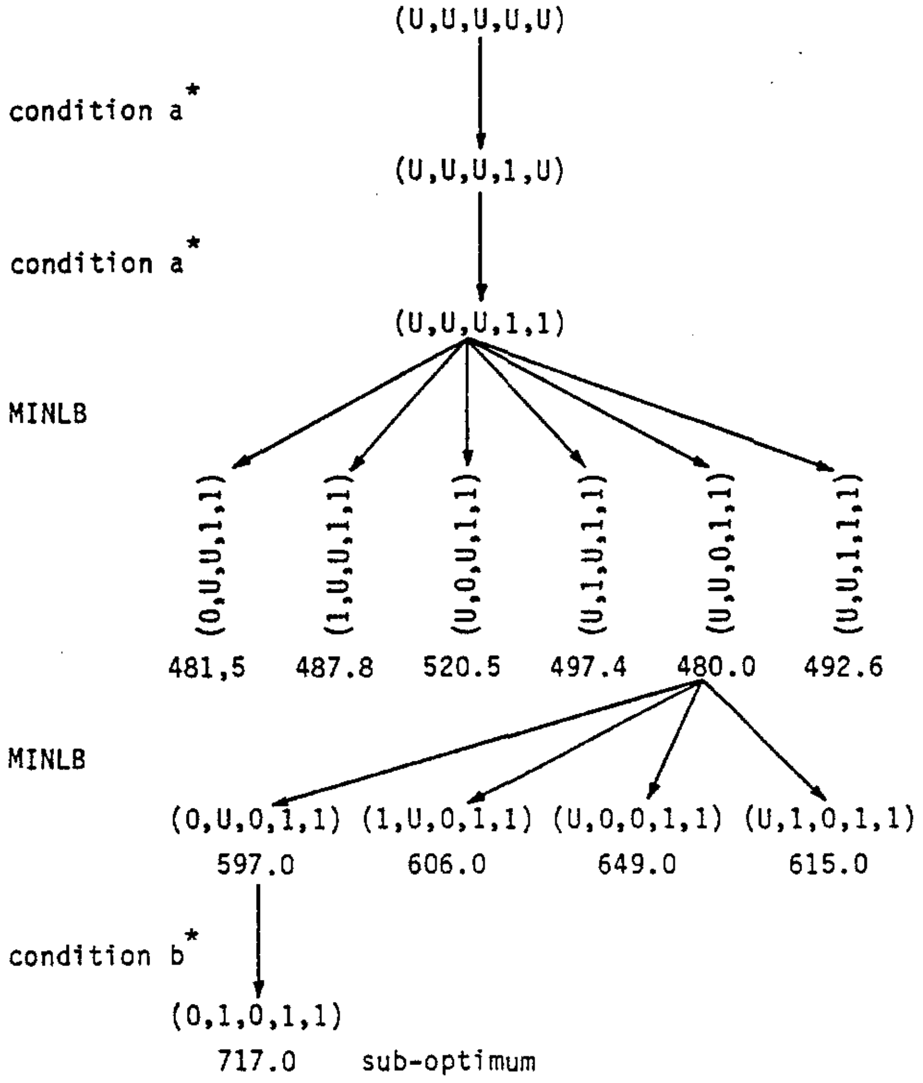
$$U = [U_i] = [\ 2\ 3\ 4\ 6\ 8]$$

$$F = [F_i] = [\ 0\ 0\ 0\ 0\ 0]$$

and

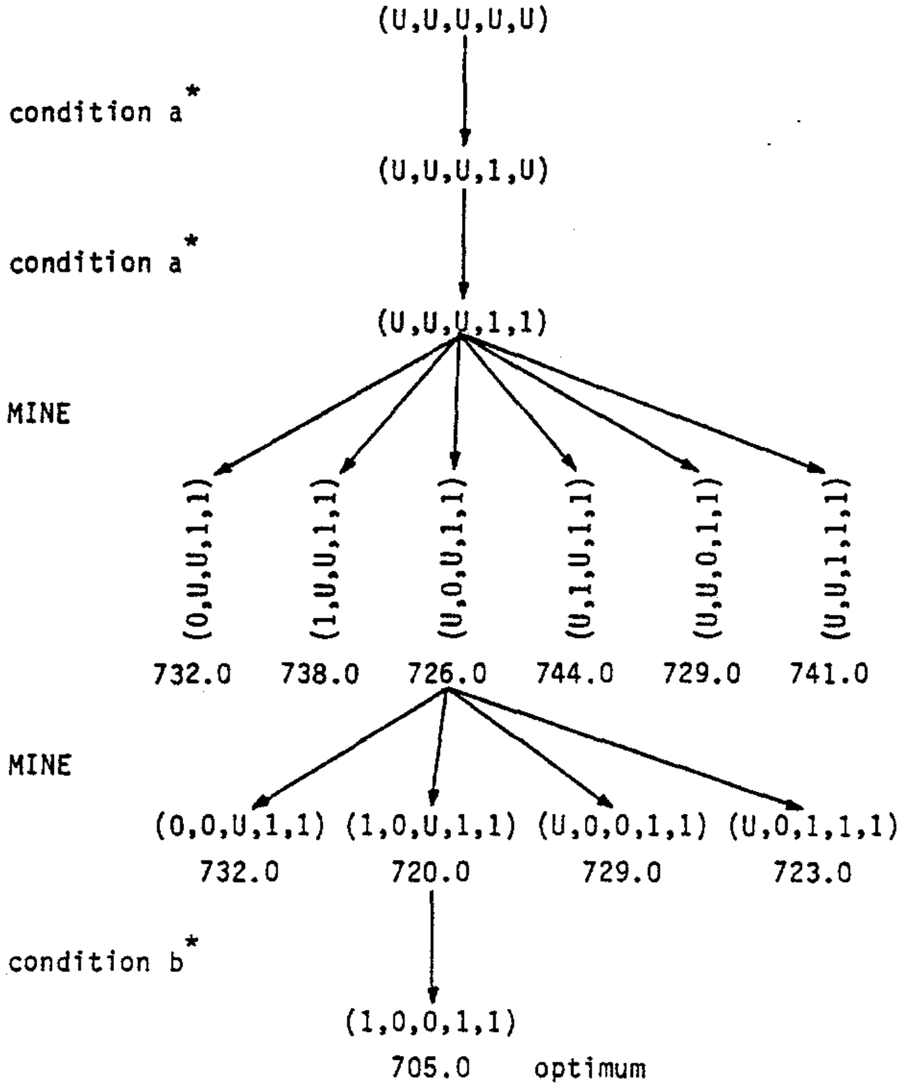
$$G = [G_i] = [\ 168\ 180\ 174\ 126\ 123].$$

By enumerating the 2^5-1 possible allocations, it is found that a copy of the file should be allocated to node 1, 4 and 5 giving a cost of 705. The steps for the four possible variations of the algorithm are shown in Figures 3.2a, 3.2b, 3,2c and 3,2d respectively. It is seen that two of these variations give the optimal solution.



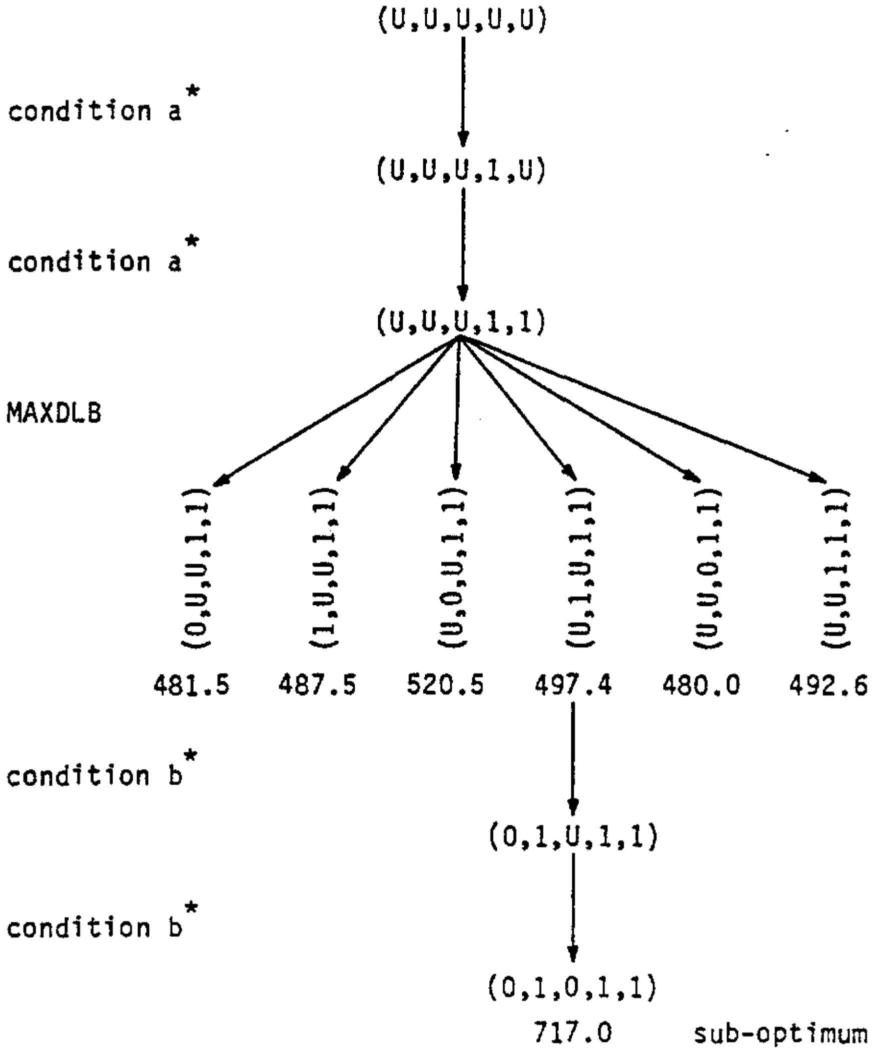
* see Table 3.5

Figure 3.2a Evaluation of Casey's 5 node Example using MINLB
(U indicates that the node is un-assigned)



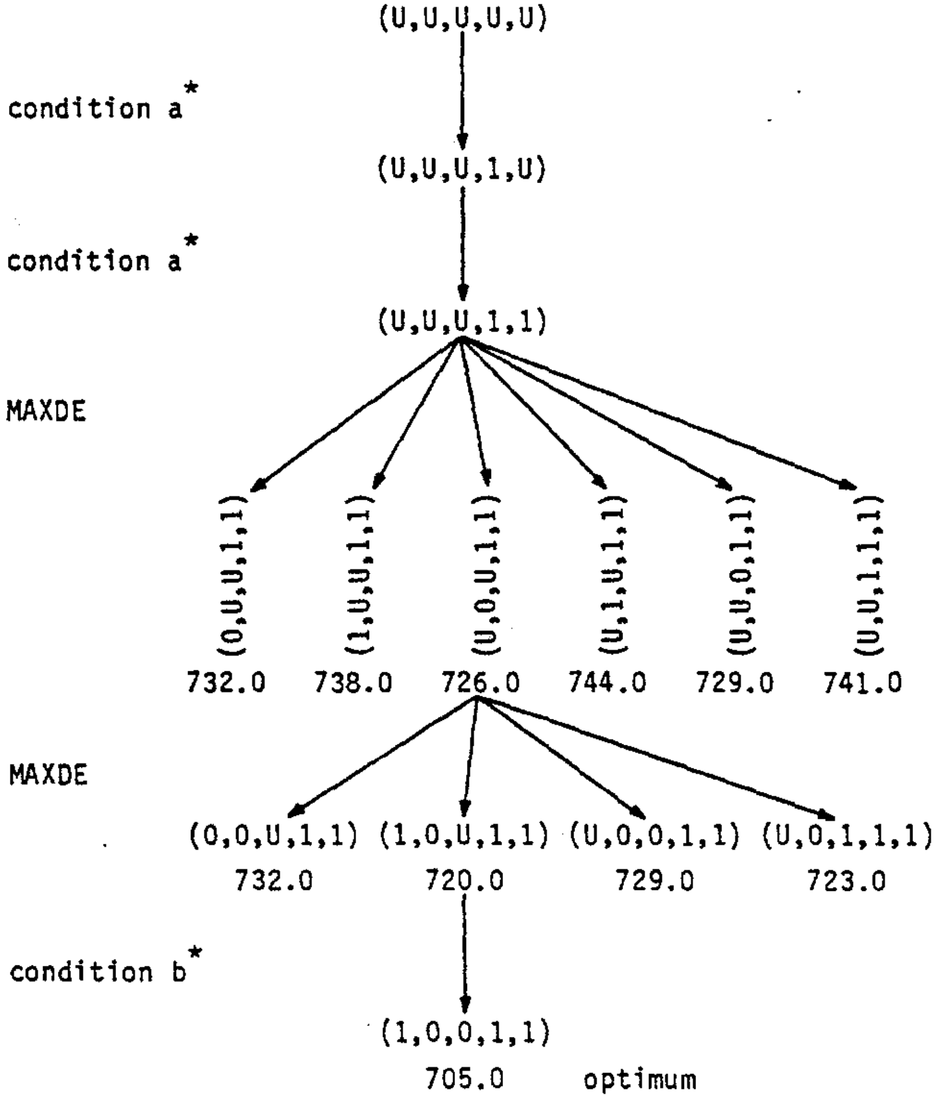
* see Table 3.5

Figure 3.2b Evaluation of Casey's 5 node Example using MINE
(U indicates that the node is un-assigned)



* see Table 3.5

Figure 3.2c Evaluation of Casey's 5 node Example using MAXDLB
(U indicates that the node is un-assigned)



* see Table 3.5

Figure 3.2d Evaluation of Casey's 5 node Example using MAXDE
(U indicates that the node is un-assigned)

The algorithm is evaluated by applying it on the published examples in the FAP and the SCWLP[5]. The optimal solutions for these examples have been established in the literature. The deviation of the heuristic solutions from the optimal solutions can be used as an indication of the "goodness" of the heuristic. The heuristic is also compared against the add-drop algorithm of Keuhn and Hamburger [KEU63][6]. The evaluation results are shown in Table 3.6. The four proposed variations of the heuristic are all polynomial algorithms and each has a complexity of $O(n^4)$ (the same as the add-drop algorithm). The execution times on the CDC 6400 are shown in Table 3.7. It is seen from Tables 3.6 and 3.7 that the algorithm MINLB gives the best results and has an execution time very small as compared with other algorithms. In fact, algorithm MINLB obtains the optimal solutions more often than the add-drop algorithm in general, but the worst case behavior seems to be worse than the add-drop algorithm and the execution times are longer because the algorithm is more complex. On the other hand, algorithm MAXDLB produces more optimal solutions than algorithm MINLB, but its worst case behavior seems to be worse. Algorithms MINE and MAXDE are much worse than algorithms MINLB and MAXDLB. Improvements can be obtained if we use the estimated lower bound (by estimating the mean and the standard deviation and making an assumption of normal distribution), but the complexity of the algorithm will become $O(n^5)$ and it takes too long to produce a solution for any of these problems (> 600 seconds). However, we can still improve the heuristic solution by combining the results of the add-drop algorithm, the MINE algorithm and the MAXDLB algorithm. In this case, over 80% of the problems will have optimal assignments and the complexity of the combined algorithm is still $O(n^4)$.

[5] The first six sets of problems are taken from [CAS72]. Problems 7 to 18 are taken from [KEU63] and problems 19 to 22 are taken from problem 7 of [SA 69, p. 1013].

[6] Instead of directly using Keuhn and Hamburger's add-drop algorithm, which selects only 5 warehouse sites to be evaluated in each cycle, the add-drop algorithm used here allows for all the unassigned warehouse sites to be taken into consideration.

Table 3.6 % Deviations of File Allocation Heuristic from Optimal Solutions

Prob.	Optimum Sol.	Add-Drop	MINLB	MINE	MAXDLB	MAXDE	Comments	
1	117596	0	0	0	8.43	0	α=0.1	Casey's 19
2	188738	0.03	0.31	0.31	0.31	0.31	α=0.2	node file
3	242581	0	0	0.66	0	0.66	α=0.3	allocation
4	291790	0	1.39	0	1.39	0	α=0.4	problem
5	431720	0	0	0	0	0	α=1.0	[CAS72]
6	705	0.85	1.70	0	1.70	0	Casey's 5 no-	de ex. [CAS72]
7	796848	0.11	0	0.78	0	0.78	Factory	keuhn and
8	854704	0.15	0.09	0.89	0	0.89	at Ind-	Hamburger's
9	893782	0.14	0	0.71	0	0.71	ianapolis	24 ware-
10	928942	0	0.61	0.94	1.49	0.99		houses, 50
11	1092916	0.08	0	0.13	0.10	0.13	Factory	customers
12	1145923	0.13	0	0.22	0	0.22	at Jack-	warehouse
13	1188241	0.13	0	1.37	0	1.37	sonville	location
14	1244991	0.22	0.22	2.49	0	1.67		problem
15	614548	0.14	0	0.90	0	0.90	Factory	[KEU63]
16	659983	0	0.12	0.80	0	0.80	at Balt-	
17	690746	0.03	0	0.74	0	0.74	imore and	
18	724886	0	0	0.42	0	0.49	Ind'polis	
19	806145	0	0	0.88	0	0.38	Factory at	Problem 7
20	870792	0.15	0	0.67	0	0.67	Ind'polis,	of Sa
21	919994	0.11	0	1.46	0	0.44	but not	[SA 69]
22	970446	0	0.42	1.73	1.36	0.67	warehouse	
	mean	0.10	0.22	0.73	0.67	0.58		
	std.dev.	0.18	0.46	0.62	1.83	0.44		

Table 3.7 Execution time of Heuristic in seconds on the CDC 6400

Prob.	Add-Drop	MINLB	MINE	MAXDLB	MAXDE	Comments	
1	0.57	11.45	22.79	11.42	103.71	α=0.1	Casey's 19
2	0.43	11.59	23.57	11.66	105.57	α=0.2	node file
3	0.43	11.77	23.60	11.76	105.50	α=0.3	allocation
4	0.43	11.80	23.48	11.79	105.26	α=0.4	problem
5	0.29	11.80	23.80	11.84	105.10	α=1.0	[CAS72]
6	0.04	0.08	0.09	0.06	0.24	Casey's 5 no-	de ex. [CAS72]
7	11.46	8.08	11.85	8.29	26.41	Factory	keuhn and
8	9.36	13.55	13.52	11.23	35.73	at Ind-	Hamburger's
9	5.34	20.89	13.91	20.99	37.61	ianapolis	24 ware-
10	3.61	8.48	8.29	8.50	21.42		houses, 50
11	12.08	6.40	9.13	6.39	17.74	Factory	customers
12	8.62	12.66	12.64	12.71	30.62	at Jack-	warehouse
13	7.82	22.16	21.26	22.83	62.03	sonville	location
14	7.02	40.18	33.47	40.33	112.93		problem
15	9.75	5.48	12.44	5.50	25.35	Factory	[KEU63]
16	7.33	5.49	4.37	4.64	7.90	at Balt-	
17	5.58	6.82	7.01	6.84	17.25	imore and	
18	3.79	2.66	3.75	2.68	7.26	Ind'polis	
19	12.24	4.94	9.16	4.95	16.04	Factory at	Problem 7
20	9.02	13.63	13.81	11.29	34.39	Ind'polis,	of Sa
21	6.76	23.27	22.05	20.97	67.74	but not	[SA 69]
22	5.94	22.79	28.81	22.02	73.40	warehouse	
mean	5.81	12.54	15.58	12.21	50.87		
std.dev.	4.11	8.92	8.83	8.84	39.27		

We have presented in this section a heuristic which can be used to obtain a file assignment with a value very close to the optimal solution. We show in the next section, that by including the migration cost into the cost function, the above heuristic is also applicable. Further, we prove some conditions for file migration on a DDB.

3.9 *DFAP - THE MIGRATION OF FILES ON A DCS*

The model that we have discussed so far assumes that the access and the update rates at each node do not vary with time. The query load (Q_j) and update load (U_j) defined in Table 3.3 are actually defined for a period of finite length. If they remain constant for every period, then the placements of files determined initially will remain static. However, it is generally true that the access and the update rates are time-varying. For example, a DCS which covers large geographic areas usually experiences different query and update rates at different parts of the system due to the different time zones in different geographic regions. It would be beneficial if the time varying characteristics of the query and the update rates are taken into account in the placements of files on the DCS.

We assume in the following discussion that time is divided into periods and the file assignments remain static within the periods. The length of each period may not be identical. The shorter the period, the more adaptive the system would be to the time-varying retrieval and update rates, but the higher would be the costs of migration which include the relocation costs and the costs of executing the file assignment algorithm. The selection of the period length is therefore very application dependent and is driven by the rate of change of the query rates and the costs of migration. It is also difficult to estimate the query rates precisely ahead of time. We therefore assume that the query rates are estimated dynamically at the beginning of each period. This may be done by using some type of working set algorithm [DEN70] which estimates the query rates based on the rate of change of the query rates in the previous periods. With this assumption, it is possible to optimize the file allocations of each period independently and is not necessary to use dynamic programming to optimize the allocations for all the periods as done by Levin [LEV74] and Sweenly and Tatham [SWE76].

There are two approaches to migrate files on a DCS:

1. *Apply stored decisions dynamically whenever restructuring is needed.*

   In such an approach, the decisions of how to restructure the file system based on the dynamic state of the system is computed beforehand. At the beginning of each period, it involves only a search of the appropriate migrations to be taken. This type of stored decision approach is very efficient because it is essentially a table look-up. However, the abundance of states usually prohibit the application of such an approach. Further, in order to store the decisions, it is necessary to find a convex hull to an n-dimensional region where n is the number of nodes in the system. The number of points on this convex hull is of the order k^n where k>1. Present algorithms to find the equation of a convex hull in four dimensional regions have an expected behavior of $O(m^2)$ where m is the number of points on the hull [BEN77] and algorithms for higher dimensions do not exist. Therefore it is unlikely that a general stored decision algorithm can be found at this time for file migration. However, by utilizing some special structure of the problem, it may be possible to find a feasible solution. This approach has been taken in communication and control systems, e.g. [CHU76, RUD77] and can be a useful and efficient heuristic if optimality requirements can be relaxed.

2. *Apply static file assignment algorithm dynamically whenever restructuring is needed.*

   This is the approach taken by most people and is the approach taken here. The disadvantages about this approach is the complexity of the optimal algorithm. However, by using a good heuristic, close to optimal results can still be obtained.

In the remainder of this section, we formulate the file migration problem for each period and show that the costs of file migration can be included into the fixed cost of the system. We define the following symbols in addition to the symbols defined in Table 3.3.

T = current period of consideration;

$S_{j,k}^{T}$ = cost of communication of one query unit from j to k in period T;

$M_{j,k}^{T}$ = cost of communication of one update unit from j to k in period T;

$N_{j,k}^{T}$ = cost of moving a copy of file a from node j to node k in period T;

F_{k}^{T} = storage cost of file at k per unit time in period T;

Q_{j}^{T} = query load originating at node j in period T;

U_{j}^{T} = update load originating at node j in period T;

C_{op}^{T} = estimated cost of running the file placement heuristic in period T;

I_{T} = index set of nodes with a copy of the file in period T;

I_{T-1} = index set of nodes with a copy of the file in period T-1.

By defining the control variable Y_j with respect to the period of consideration, we have:

$$Y_j^T = \begin{cases} 0 & j \not\in I_T \\ 1 & j \in I_T \end{cases}$$

The access and the update costs are the same as in Equations 3.1 and 3.2 except that the costs per unit time are defined for period T specifically. Further, there is an additional component of the costs, the migration cost.

$$File\ Migration\ cost = \sum_{k=1}^{n} Y_k^T \min_{j \in I_{T-1}} N_{j,k}^T$$

That is, if node k does not have a copy of the file in period T and it is necessary to migrate a copy of the file to node k, then a copy of the file is migrated from the nearest node in the assignments of period T-1. It is easily seen that optimization problem 3.1 can be written in the original form with only a change in the values of G_k (Eq. 3.2).

min

$$C(I) = \sum_{j=1}^{n} Q_j^T \min_{k \in I_T} S_{j,k}^T + \sum_{k=1}^{n} G_k^T Y_k^T \tag{3.8}$$

subject to

$$Y_k^T = 0\ or\ 1\ (integer) \qquad k=1, \ldots, n$$

and

$$G_k^T = F_k^T + \sum_{j=1}^{n} U_j^T M_{j,k}^T + \min_{j \in I_{T-1}} N_{j,k}^T \tag{3.9}$$

The importance of the above formulation is that the static file assignment algorithms developed in the literature and the file assignment heuristic described in section 3.8 are still applicable to solve the file assignment problem in each period although migration costs have been included in the formulation. Therefore, at the beginning of each period, it is only necessary to determine Q_j^T, U_j^T, and G_j^T for all j∈{1,...,n} and the static file assignment algorithm can then be applied.

3.10 *CONDITIONS TO REDUCE THE COMPLEXITY OF THE DFAP*

In this section, we want to establish some general theorems on the DFAP which will aid in simplifying the problem. Specifically, we want to show the NP-completeness of the problem of selecting the migration points and to find an upper bound on the number of file migrations in period T.

3.10.1 *The Problem of Selecting the Times for Migration is NP- Complete*

Since Eswaran has shown that the FAP is NP-complete [ESW74], the DFAP, which is a general case of the FAP, is also NP-complete. However, we want to show that the problem of selecting the points of migration in a multi-period length of time is also NP-complete. This means that we have to exhaustively enumerate over all the possibilities before we can decide when to initiate a file migration. We achieve this by reducing the knapsack problem to the problem of selecting the points of migration.

Knapsack Problem [KAR72]

Input: $(a_1, a_2, \ldots, a_r, b) \in Z^{n+1}$; Z = set of integers;

Property: $\sum a_j x_j = b$ has a 0-1 solution for x_j.

Problem of selecting the migration points - feasibility form

During a time period [0,t], at what points of time should migrations be initiated so that the total operating cost = B

We assume that the query rates are changing with time and that migrations can only be initiated at fixed discrete times, $t_1, t_2, \ldots, t_k$ within the period [0, t]. The last assumption is made because computer operations are governed by a clock which is discrete.

THEOREM 3.2

The problem of selecting the migration points is NP-complete

Proof

First, we want to show that the problem $\in$ NP. A non-deterministic Turing machine can guess the set of times at which the files in the system are to be migrated and therefore the problem $\in$ NP.

Second, we have to show that the satisfiability problem (SAT) is reducible to this problem (SAT $\propto$ the problem of selecting the migration points). We can do this by showing that the knapsack problem $\propto$ this problem because SAT $\propto$ knapsack and by transitivity, SAT $\propto$ this problem. Given an instance of the knapsack problem, we can construct (in polynomial time), an instance of the problem of selecting the migration points as follows:

Let

$$x_i = \begin{cases} 0 & \text{if } \textit{no migration is initiated at } t_i \\ 1 & \textit{otherwise} \end{cases}$$

a_i = the costs of migration at time t_i. (The costs are not the same at different t_i's because the costs may be discounted to time t_0, or different costs may be associated with different times).

B = b.

There are no other costs associated with the operation of the system.

The knapsack problem is therefore reducible to the problem of selecting the migration points. Since the knapsack problem is NP-complete, hence, we have proved the theorem.

Q.E.D.

After establishing that the problem of selecting the migration points is NP-complete, we are left with two alternatives: (1) exhaustively check the 2^k possibilities of whether to migrate at the k discrete times within the period [0, t]; or (2) establish some criteria for migration. The first alternative has been taken by Levin [LEV74] and Sweenly and Tatham [SWE76]. We investigate the second alternative here.

3.10.2 *Criteria for Initiating a Migration*

We want to establish in this section some criteria under which migration should be carried out. First, we want to find the maximum number of necessary file movements in any migration.

Lemma 3.1

Given the allocations of the multiple copies of a particular file, the maximum number of file movements needed is n-1.

Proof

A file movement is needed for node i whenever $Y_i = 0$ before the migration and $Y_i = 1$ after the migration. Under no other cases should there be a file movement. It is also assumed that there is at least a copy of the file on the system. Therefore, the maximum number of file movements occur when there are n-1 nodes without a copy before the migration and these n-1 nodes have copies after the migration.

Q.E.D.

Given an allocation in period T, we are interested in finding a lower bound and an upper bound on the costs of p file movements, p = 1, ..., n-1 in period T+1.

Recall that:

$K_0^T = \{j : Y_j^T = 0\}$

$K_1^T = \{j : Y_j^T = 1\}$

and assume that all the nodes have been assigned, i.e. $K_2^T = \phi$.

Let

$C_L^p (C_U^p)$ = lower (upper) bound on the costs of p file movements, $1 \leq p \leq n - |K_1|$.

The following algorithm finds C_L^p, C_U^p.

Algorithm 3.2 - To find the Lower and the Upper Bounds on the Costs of p file movements:

```
1. $C_L^p \leftarrow 0$;    $C_U^p \leftarrow 0$;
   $K_{0,L}^{T+1} \leftarrow K_0^T$;  $K_{1,L}^{T+1} \leftarrow K_1^T$;
   $K_{0,U}^{T+1} \leftarrow K_0^T$;  $K_{1,U}^{T+1} \leftarrow K_1^T$;
```

2. Do Steps 3 and 4 p times;

3. $C_L^p \leftarrow C_L^p + \min\limits_{\substack{j\in K_{1,L}^T \\ k\in K_{0,L}^{T+1}}} N_{j,k}^T$;

$K_{1,L}^{T+1} \leftarrow K_{1,L}^{T+1}\cup\{k\},\ K_{0,L}^{T+1} \leftarrow K_{0,L}^{T+1}-\{k\}$;

4. $C_U^p \leftarrow C_U^p + \max\limits_{k\in K_{0,U}^{T+1}} \min\limits_{j\in K_{1,U}^T} N_{j,k}^T$;

$K_{1,U}^{T+1} \leftarrow K_{1,U}^{T+1}\cup\{k\},\quad K_{0,U}^{T+1} \leftarrow K_{0,U}^{T+1}-\{k\}$.

Note that $C_L^p \leqq C_L^{p+1}$ because all the costs involved are positive.

Having established the lower and the upper bounds on the cost of p file movements, we want to compute the change in total system costs due to a perturbation in the access rate. When the change in total system cost is greater than a threshold, a file migration is necessary. The change in total system cost is given partially by the following theorem.

THEOREM 3.3

Let

$R_j^T = Q_j^T + U_j^T$;

$r_j^T = Q_j^T / R_j^T$;

$R_j^{T+1} = R_j^T + \omega_j^{T+1}$ where ω_j^{T+1} is the perturbation in the total number of accesses in period T+1 at node j and is proportionally divided between retrievals and updates;

$$C_\omega^T = \sum_j r_j^T \omega_j^{T+1} \min_{k\in I_T} S_{j,k}^T + \sum_{j,k} (1-r_j^T)\omega_j^{T+1} M_{j,k}^T Y_k^T.$$

= Cost increase due to the perturbation.

(a) If $\forall j$, $\omega_j^{T+1} \geqq 0$, then the upper bound of file movements that can be made on the system is p' where

$$p' = \min\{p : C_L^p + C_{op}^T > C_\omega^T\} - 1 \tag{3.10}$$

(b) If $\exists$ j, $\omega_j^{T+1} < 0$, then the lower bound of file movements that can be made on the system is 0.

Proof

Let

$Y^T = (Y_1^T, Y_2^T, \ldots, Y_n^T)$ be the original optimal state of allocation in period T;

$Y^{T+1} = (Y_1^{T+1}, Y_2^{T+1}, \ldots, Y_n^{T+1})$ be the state of allocation after the perturbation in period T+1;

$C(Y^T)[C(Y^{T+1})]$ = cost of operation at state $Y^T(Y^{T+1})$.

We want to show:

(1) C_ω^T is an upper bound in the cost increase due to ω_j^{T+1} if $\forall j$, $\omega_j^{T+1} \geqq 0$;

(2) C_ω^T is a lower bound in the cost saving due to ω_j^{T+1} if $\forall j$, $\omega_j^{T+1} \leqq 0$;

(3) if $\exists |$ i,j s.t. $\omega_i^{T+1} > 0$, $\omega_j^{T+1} < 0$ and $C_\omega^T \leqq 0$, then C_ω^T is a lower bound in the cost saving;

(4) if $\exists |$ i,j s.t. $\omega_i^{T+1} > 0$, $\omega_j^{T+1} < 0$ and $C_\omega^T > 0$, then the lower bound in the cost saving is 0.

To prove:

(1) We observe that $\exists | Y^{T+1}$ s.t.

$$C(Y^T) \leqq C(Y^{T+1}) - C_\omega^{T+1} \leqq C(Y^{T+1}) \leqq C(Y^T) + C_\omega^T$$

where

$$C_\omega^{T+1} = \sum_j r_j^T \omega_j^{T+1} \min_{k \in I_{T+1}} S_{j,k}^T + \sum_{j,k} (1 - r_j^T) \omega_j^{T+1} M_{j,k}^T Y_k^{T+1}$$

The first inequality can be proved by contradiction. If $C(Y^T) > C(Y^{T+1}) - C_\omega^{T+1}$, this means that $C(Y^{T+1}) - C_\omega^{T+1}$, which is the cost of operation at state Y^{T+1} without the cost of the perturbation, has a lower cost than state Y^T. This implies that state Y^T cannot be the optimal state of allocation which contradicts the original assumption.

For the second inequality, $C(Y^{T+1}) \geqq C(Y^{T+1}) - C_\omega^{T+1}$, we observe that $C_\omega^{T+1} \geqq 0$ if all $\omega_i^{T+1} \geqq 0$.

The third inequality, $C(Y^T)+C_\omega^T \geqq C(Y^{T+1})$, can be proved by contradiction. If $C(Y^T)+C_\omega^T < C(Y^{T+1})$, then it is not necessary to re-organize the data base to state Y^{T+1} where the cost of operation is higher than the cost involved without the re-organization.

Therefore, $0 \leqq C(Y^{T+1})-C(Y^T) \leqq C_\omega^T$ and C_ω^T is the upper bound in the cost increase.

(2) We observe that:

$$C(Y^{T+1})-C_\omega^{T+1} \geqq C(Y^T) \geqq C(Y^T)+C_\omega^T \geqq C(Y^{T+1})$$

The proof is exactly the same as part (1) with an inter-change of Y^T and Y^{T+1}.

Therefore $C(Y^T)-C(Y^{T+1}) \geqq -C_\omega^T$ and C_ω^T represents a lower bound in the cost savings.

(3) We observe a similar condition as part (2).

$$C(Y^{T+1})-C_\omega^{T+1} \geqq C(Y^T) \geqq C(Y^T)+C_\omega^T \geqq C(Y^{T+1})$$

Therefore, $C(Y^T)-C(Y^{T+1}) \geqq -C_\omega^T$.

(4) We can only establish a weaker condition in this case:

$$C(Y^T) \leqq C(Y^{T+1})-C_\omega^{T+1}$$

$$C(Y^{T+1}) \leqq C(Y^T)+C_\omega^T$$

These two inequalities can be proved similarly as before. We cannot prove any relation between $C(Y^{T+1})-C_\omega^{T+1}$ and $C(Y^{T+1})$ because C_ω^{T+1} may be $\geqq 0$ or $<$ 0.

In summary, we have proved for case

(1) $0 \leqq C(Y^T)-C(Y^{T+1})+C_\omega^T \leqq C_\omega^T$

(2),(3),(4) $0 \leqq C(Y^T)-C(Y^{T+1})+C_\omega^T$.

We can now prove the theorem.

(a) We observe that $C_L^p+C_{op}^T$ is a lower bound on the costs of running the optimization program and initiating p file movements, so in order for the reconfiguration to be cost-effective, we must have

$$C_\omega^T \geqq C(Y^T)-C(Y^{T+1})+C_\omega^T \geqq C_L^p+C_{op}^T$$

The upper bound on the number of file movements is

$$\boldsymbol{p} = \min\{p : C_L^p+C_{op}^T > C_\omega^T\}-1$$

(b) We note that the lower bound on the cost savings is $\geqq 0$, so the lower bound on the number of file movements is $\geqq 0$.

Q.E.D.

Although the above theorem does not provide us with an upper bound on the number of file movements when some or all of the ω_i^{T+1}'s are less than zero, we can still find an upper bound on the number of file movements if we can establish a lower bound on the costs of operation for the perturbated state of accesses. In these cases, i.e., when some $\omega_i^{T+1}<0$, we can estimate $\overline{C}(Y^{T+1})$, the lower bound on the optimal cost of operation after the perturbation without taking into account the cost of migration. Then

$$C_\Omega^T = C(Y^T) + C_\omega^T - \overline{C}(Y^{T+1}) \geqq 0$$

is an upper bound on the cost savings due to migration.

The maximum number of file movements is therefore

$$\boldsymbol{p} = \min\{p : C_L^p+C_{op}^T > C_\Omega^T\}-1 \qquad (3.11)$$

where

$$C_\Omega^T = \begin{cases} C_\omega^T & \text{if } \omega_j^{T+1} \geqq 0 \quad \forall j \in \{1,\dots,n\} \\ C(Y^T)+C_\omega^T-\overline{C}(Y^{T+1}) & \text{if } \exists j \in \{1,\dots,n\} \ s.t. \ \omega_j^{T+1}<0 \end{cases}$$

The problem that remains is to compute the lower bound $\overline{C}(Y^{T+1})$. This can be done by solving the optimization problem (3.1) without the integrality constraints (see Appendix B). Theorem 3.3 therefore establishes the basis for the initiation of a file migration on the DDB. It has also taken into account the cost of running the optimization program for the FAP. It indicates that when it is very expensive to run the optimization program for the FAP, it will not be cost effective to do file migration.

3.11 *CONCLUSION*

In this chapter, we have investigated some important properties and solution algorithms for the File Allocation Problem and the Dynamic File Allocation Problem. First, we have proved the isomorphism between the (dynamic) file allocation problem and the single commodity (dynamic) warehouse location problem. Based on this property, we have found that many techniques developed for both problems are inter-changeable. Among these are algorithms developed in the warehouse location problem, such as the add-drop algorithm, the branch and bound algorithms, the probabilistic branch and bound algorithm, the integer programming technique, the steepest ascent algorithm and the dynamic programming methods. These algorithms can be applied to solve the (dynamic) file allocation problem. On the other hand, there are algorithms developed in the file allocation problem which can be used to solve the warehouse location problem. These include the hyper-cube technique, the clustering technique, the dynamic programming methods and the max-flow min-cut network flow technique. Further, we have found that some techniques developed for one problem match very closely techniques developed for the other problem. This is shown by the fact that Grapa and Belford's conditions for locating a copy of the file at a node [GRA77b] are weaker than the conditions derived by Efroymson and Ray for opening or closing a warehouse [EFR66]. This implies that by using the stronger conditions of Efroymson and Ray, more nodes can be assigned initially to have or not to have a copy of the file. Another example is shown in the similarity in the dynamic programming technique applied by Levin to solve the dynamic file allocation problem [LEV74] and by Sweenly and Tatham to solve the dynamic warehouse location problem [SWE76]. The last example is shown in the hypercube technique which has been developed at different times by Casey [CAS72], Levin and Morgan [LEV74, MOR77] and Alcouffe and Muratet [ALC76]. We conclude that these two problems can be studied in an integrated fashion in the future.

Second, we have developed a heuristic to solve the file allocation problem. This heuristic uses the add-drop principle and different criteria on selection are compared. It is found that a combination of these criteria, together with the add-drop algorithm, is very promising and gives solutions very close to the optimum based on sample problems published in both the file allocation problem and the warehouse location problem.

Lastly, we have studied some aspects of the file migration problem. It is shown that the problem of deciding when to migrate the files is NP-complete. This means that it is likely that an exhaustive enumeration is necessary before an optimal migration sequence can be found. We have also formulated the migration problem and have shown that the migration costs can be incorporated into the fixed cost of the system. This implies that the file allocation heuristic developed in this chapter can be applied to solve the file migration problem without special considerations for the costs of migration. Finally, we have developed a threshold to indicate when migration should be carried out.

4 *TASK SCHEDULING ON DISTRIBUTED COMPUTER SYSTEMS*

4.1 *INTRODUCTION*

In the previous chapters, we have addressed the optimization problems of data management on the query and the file level. The operations to be performed on the query and the file level is a conglomerate of tasks, each of which may require the use of a different resource for a different amount of time. In this chapter, we address the problem of the task scheduling on DCS's so that the hardware can be efficiently utilized and the requirements can be satisfied.

Although one of the motivations for the development of DCS's is the declining hardware costs, and therefore efficient hardware utilization is not as important a problem as in early computer systems, the problem of task scheduling is still an important topic of research because the parallel resources are more difficult to coordinate and there are other constraints on the system which must be satisfied, e.g. deadlines, response time, etc. Further, the advantages of using parallel hardware is lost if the improvement over a conventional uni-processor system is small. It is the goal of this chapter to study the problem of task scheduling on DCS's.

A *task* is defined to be a simple request which uses a resource for a finite amount of time. A request is said to be simple if no other resource is needed during the processing of this request. A complex request can always be broken down into a sequence of simple requests. A resource on a DDB can be physical, such as a communication channel, a processor, etc., or it can be logical, such as a file. The tasks are usually governed by a precedence graph so that a task cannot be processed until its predecessor has finished processing. For example, in order to handle a file request on a data base, many processes, such as receive message, create transaction, assemble reply, file storage I/O, etc., have to be activated. Another example is shown in the processing of user queries, which are directed to the different nodes on the DCS. Each of these queries may be partitioned into a set of tasks. The general precedence graph for the processing of a query which require the use of geographically distributed files are shown in Figure 4.1. On a DCS, the communication overheads, which include time to set up the communication path and the queueing delay to transmit the messages, are usually much larger than the processing overhead for a query. Therefore, the time required to process a task at a node in Figure 4.1 is usually negligible when compared with the time to pass the results over the communication subsystem. There are also other queries on the system, each of which has its own task precedence graph. There may also be precedence constraints among the precedence graphs of the different queries. The *task scheduling problem* that we are concerned with here, is to sequence the processing of tasks, subject to precedence constraints, so that some overall optimization criteria are satisfied. The criteria can be the maximum completion time of all the tasks if the objective is to maximize the throughput of the system; or it can be the sum of the completion times of all the tasks if the objective is to minimize the average response time; or it may be a combination of several optimization criteria.

We first describe a model of the DCS and state some tradeoffs which can be used to simplify the problem. We show that the problem of deterministic scheduling on this model is NP-complete. Since the problem is NP-complete, it is unlikely that a polynomial algorithm can be found. We proceed to study the problem by putting additional constraints on the model so that the problem is polynomially solvable. The resultant model we have obtained is the model for an interleaved memory. We study in detail the performance of an interleaved memory and show that the polynomial scheduling algorithm we have developed is an optimal average behavior algorithm. That is, the polynomial algorithm will have the best average performance as compared with any other polynomial algorithms. Lastly, we return to the original model and show a heuristic for the

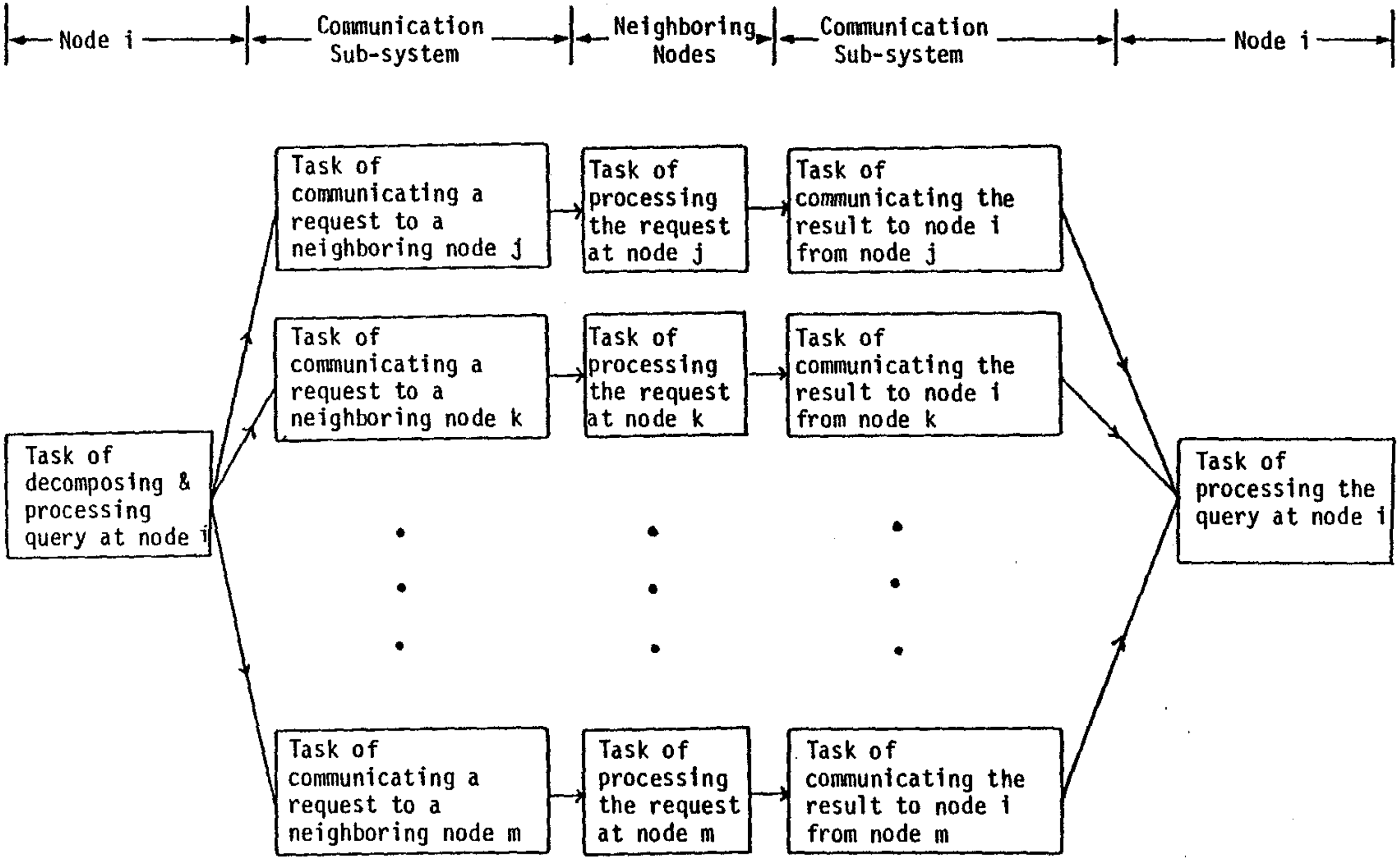

Node i
Communication Sub-system
Neighboring Nodes
Communication Sub-system
Node i
Task of decomposing & processing query at node i
Task of communicating a request to a neighboring node j
Task of processing the request at node j
Task of communicating the result to node i from node j
Task of communicating a request to a neighboring node k
Task of processing the request at node k
Task of communicating the result to node i from node k
Task of communicating a request to a neighboring node m
Task of processing the request at node m
Task of communicating the result to node i from node m
Task of processing the query at node i

Figure 4.1 Task Precedence Graph for the Processing of a Query which requires the use of Geographically Distributed Files

scheduling of tasks on the general model. Some simulation results for this heuristic are also shown.

4.2 *A MODEL FOR THE SCHEDULING OF TASKS ON DISTRIBUTED SYSTEMS*

4.2.1 *The Model*

Flynn [FLY66] has classified methods of achieving parallel operations into four classes: the single instruction, single data stream (SISD), the single instruction, multiple data stream (SIMD), the multiple instruction, single data stream (MISD), and the multiple instruction, multiple data stream (MIMD). A basic model of a computer system on a DCS for the scheduling of tasks is the SIMD model. This model is shown in Figure 4.2. The control unit may represent the CPU. The N arithmetic processors may represent the peripheral processors or the backend machines. An instruction may be a search for a particular item on the mass storage and the data streams are coming directly from the disks. Another example of a SIMD architecture is the Data Base Machine [HSI77]. On the DCS level, the DCS may be represented by a Job-Shop model in which the basic building block within the job-shop model is the SIMD model. A job-shop is a model which has been used in industrial engineering and deterministic task scheduling [GRA77a]. The characteristic of the job-shop model is that a job or a request is made up of a set of tasks, each of which may be processed on a given machine or processor for a given amount of time. A conceptual model of a DCS is shown in Figure 4.3. The graph is actually a fully connected graph in which an arrow represents an instruction stream and the corresponding return data flow. This is a more restricted model than the general job-shop model because each job or request is made up of only a set of parallel tasks. The basic model at each node is the SIMD model.

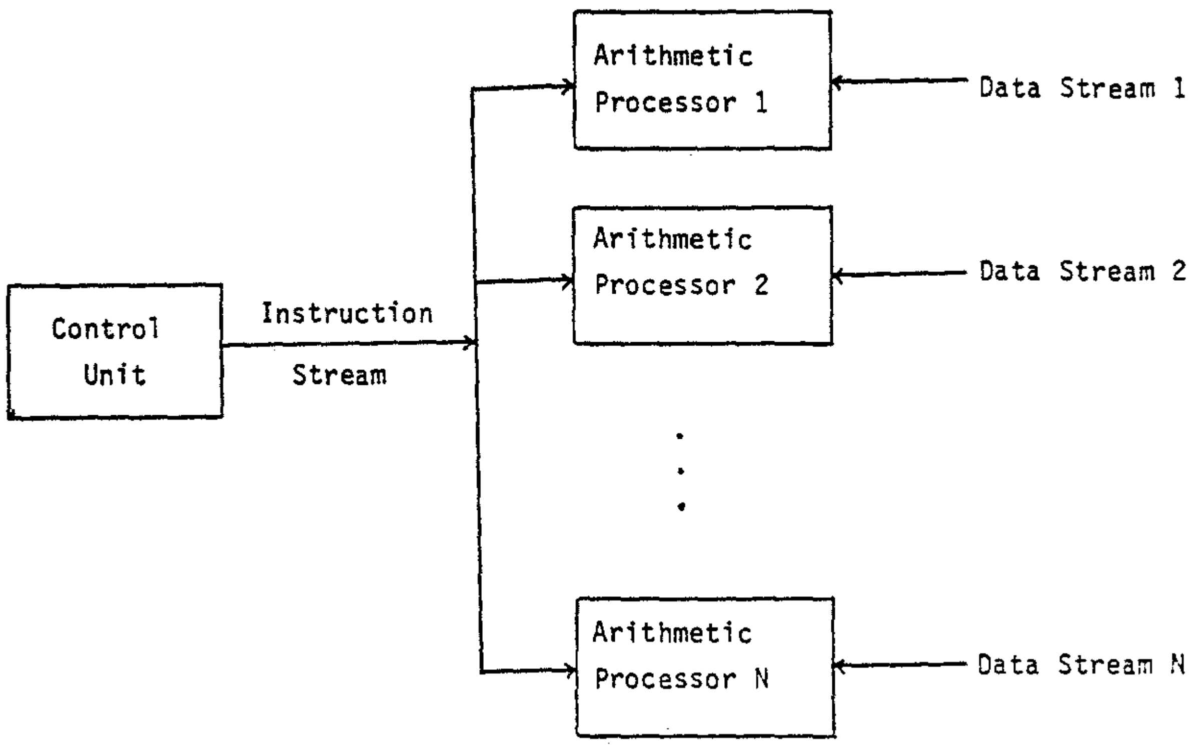


Figure 4.2 Model of an SIMD Computer System [STO75]

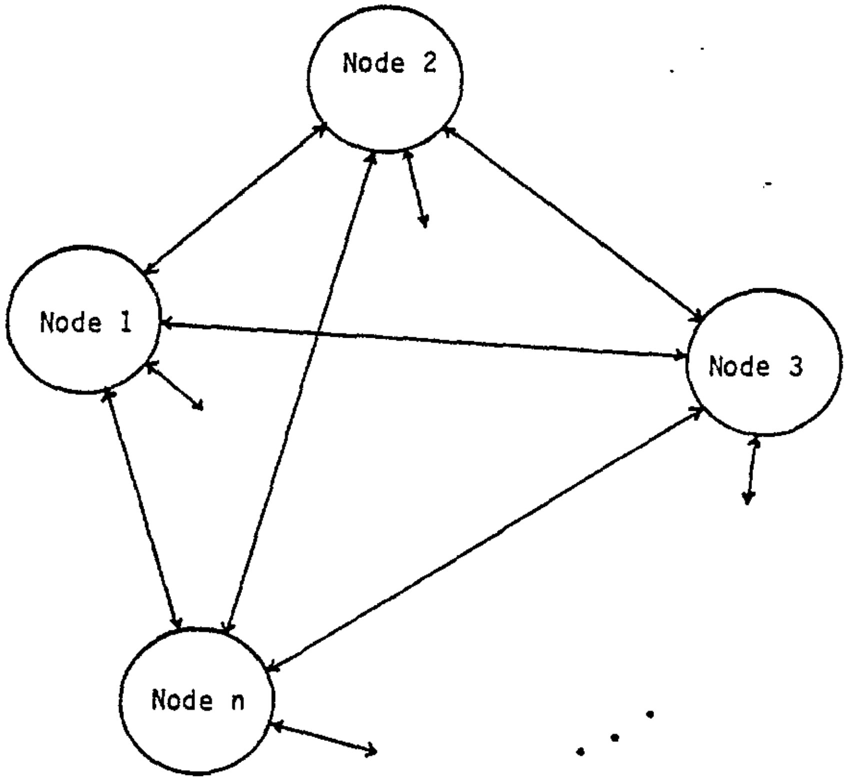


Figure 4.3 Conceptual Model of a DCS (The direction of an arrow represents the flow of an instruction stream and the corresponding return data flow)

The model we have discussed here can be more general. For example, each job or request may consist of a sequence of tasks to be scheduled on different nodes or computers instead of a set of parallel tasks to be scheduled on neighboring nodes. However, this scheduling problem can be solved only when the status of all the nodes of the DCS is known. This is possible when the scheduling is done by a centralized control and all the status changes are reported instantly to the centralized controller. In a geographically distributed DCS, the collection of global information for scheduling is usually very difficult and expensive if not impossible. Therefore the use of a more general model is usually not practical for a DCS. As a result, we have restricted to the case in which the scheduling of tasks is done by using the local information available (distributed control), that is, it is a SIMD model at each node. The restricted model to be studied is shown in the dotted box in Figure 4.4. The notations used in Figure 4.4 are:

N - number of tasks to be scheduled (it may or may not fit entirely in the buffers of M_a);

M_a - Distributor on the first stage;

$M_{b,j}$ - module or machine j on the second stage;

$P_i(M_j)$ - Processing time requirement of task i on machine M_j;

$buff\,(M_j)$ - Amount of buffers for M_j.

The task precedence graph for request i is shown in Figure 4.5. The precedence graph in Figure 4.1 falls in the class of precedence graphs we discuss here if the tasks of communicating to and from node i and the task of processing at a neighboring node are combined into a single task. We assume that the optimization criterion is to minimize the finish time of all the tasks in the system. This is generally the assumption made when the objective is to maximize the throughput of the system.

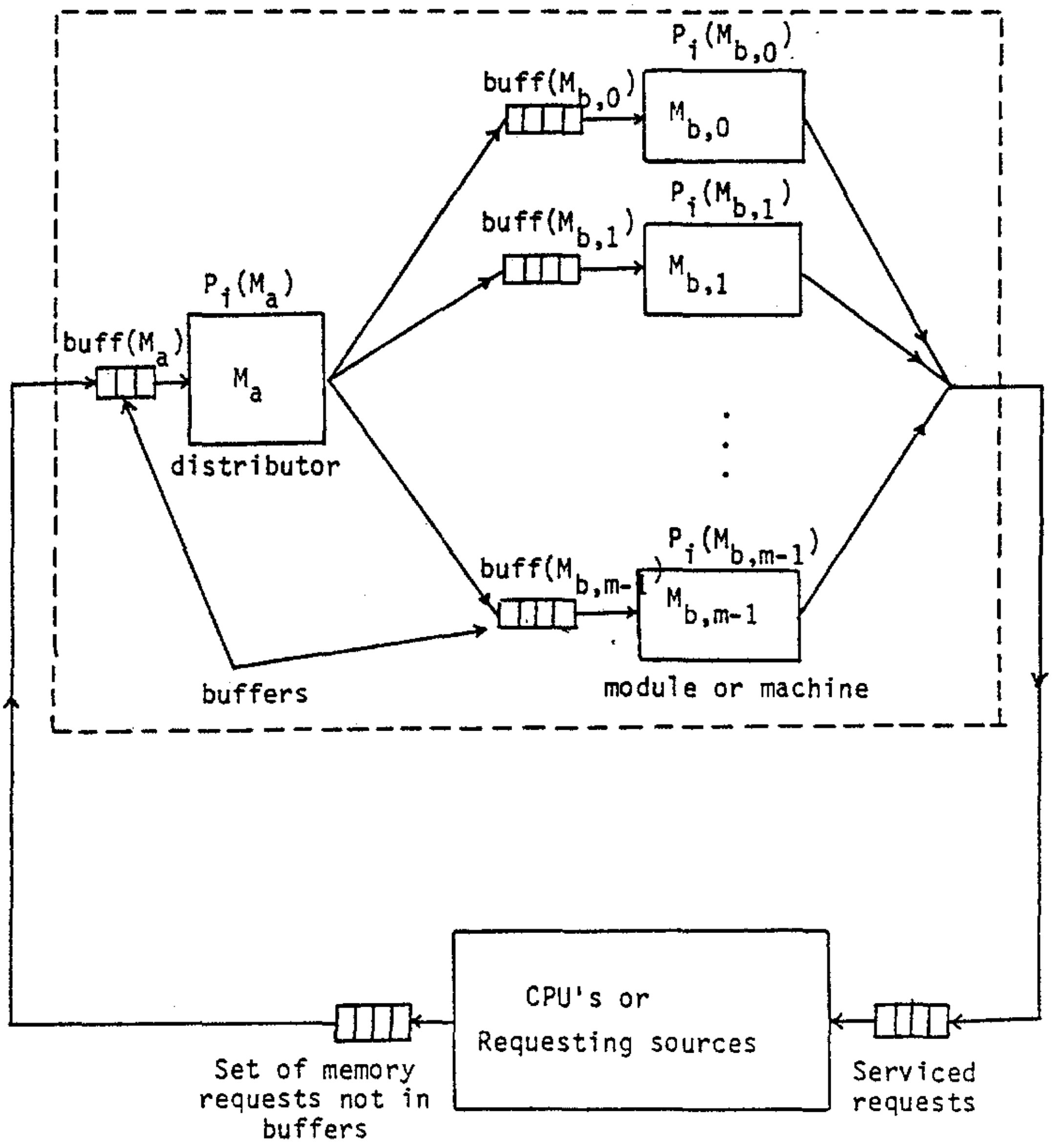


Figure 4.4 An SIMD Model for Task Scheduling on a DCS

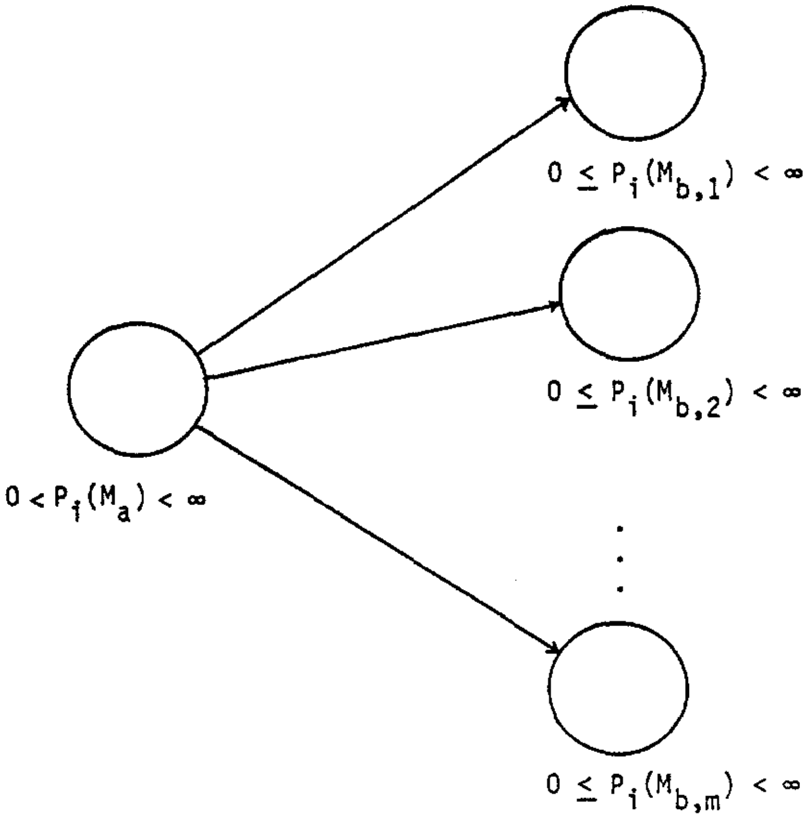

$0 \leq P_i(M_{b,1}) < \infty$
$0 \leq P_i(M_{b,2}) < \infty$
$0 < P_i(M_a) < \infty$
$0 \leq P_i(M_{b,m}) < \infty$

Figure 4.5 Precedence Graph of Tasks for Request i which can be scheduled on the SIMD Model

It is also assumed that the amount of buffers on the second stage is finite and that the amount of buffers on the first stage may be infinite, that is,

$$0 \leqq buff(M_a) \leqq \infty;$$

$$0 \leqq buff(M_{b,1}) = buff(M_{b,2}) = \cdots = buff(M_{b,m}) < \infty.$$

Further, it is assumed that there may exist precedence constraints among different requests and the tasks may not be available initially, that is, they have positive release dates. The analysis of this model is shown in Section 4.3. We now discuss some assumptions which would allow the problem to be simplified.

4.2.2 *Assumptions which allow the Task Scheduling Problem to be simplified*

Certain assumptions can be made so that the task scheduling problem can be simplified.

(1) *Processing Overheads are ignored*

The processing overheads are usually much smaller than the communication overheads and they are ignored. This assumption will eliminate many tasks in the precedence graph.

(2) *Static Algorithms are used*

Static algorithms schedule a set of tasks available at the time of scheduling and a set of tasks that are known to arrive at fixed future times. The schedule does not change during the duration of the processing of these tasks. On the other hand, dynamic algorithms are more flexible and they re-schedule all the available tasks whenever a new task comes in. The advantage of dynamic algorithms is that they allow task initiations to be dynamic and do not restrict the schedule to the order determined initially, but they have the disadvantage of larger overheads. The choice between the use of static and dynamic algorithms is system dependent. If the arrivals of requests are indeterminate, then dynamic algorithms are usually better. On the other hand, if the arrivals of requests can be determined precisely, then static algorithms should be used. In our model, we have assumed that static algorithms are used because it does not depend on the arrival process and is easier to optimize. The static algorithm developed can be used as a heuristic when the arrivals of requests are indeterminate.

(3) *Deterministic Processing Times are assumed*

The processing time for a task can be assumed to be deterministic or probabilistic. In the deterministic case, it is possible to determine the order which can best satisfy the optimization criterion. However, it is difficult to do so when the processing times of all the tasks are governed by a common distribution. Certain assumptions have to be made before an analytical evaluation is possible. The theory of scheduling developed now is mostly applicable to the deterministic case. It can be used to approximate the probabilistic case when the average or the worst case processing times are used. A lot of work has been done in flow shop and job shop scheduling, (see [GRA77a, LEN77] for a good survey) and the theory developed there can be applied to study the problem here. The algorithm developed in Section 4.5 is actually extended from Johnson's optimal polynomial algorithm for a two stage flow shop [JOH54]. On the other hand, when the processing time of a task is probabilistic, the model we have shown in Figure 4.4 is a "central server model", and a lot of work in queueing theory has been done to evaluate its performance. For example, Baskett et. al. have developed a closed form formula for the performance of a queueing network when certain conditions are satisfied [BAS75]; Sauer and Chandy have developed approximate analysis techniques for central server models [SAU75]; Chandy et. al. have studied approximate analysis techniques for general queueing networks [CHA75]. Unfortunately, when a probabilistic assumption is made on the processing time of a job, it is usually difficult to determine the order of processing which can satisfy some optimization criteria. Some work has been done in finding a service schedule which minimize expected costs [MEI77, KON68, KLI74], however, a general theory for this is still lacking. Therefore, we see that it is easier analytically to make the deterministic assumption. One other advantages about the deterministic assumption is that the difficulty of the scheduling problem can be assessed easily in most cases. NP-completeness of the problem can usually be shown or a polynomial algorithm can be found. The general task scheduling problem on DCS's using our model can be shown to be NP-complete. Under this situation, the designer has to look for good heuristics which can be executed within real time constraints. However, the evaluation of heuristics are generally difficult. Evaluation methods and techniques are typically of three kinds, analytical techniques, simulations and approximate algorithms. In analytical techniques, some simplifying assumptions about the system parameters have to be made in order for the solution to be tractable and the results obtained are usually not accurate. On the other hand, simulations are almost always expensive to run, and it is difficult to exhaust all the possible cases of the system. A third type of evaluation algorithms are approximate algorithms [WEI77]. There are two classes of these approximations, one guaranteeing a near-optimal solution always, and the other producing an optimal or a near-optimal solution "almost everywhere". These types of algorithms are still in the research stage and a unifying approach in designing algorithms of this type is still lacking. The future trend is in the direction of investigating good approximation algorithms for scheduling tasks.

By making the assumptions in this section, we have sacrificed some generality for some mathematical tractability. We hope that the results we have obtained here are still applicable (to some extent) when these assumptions are relaxed.

4.3 *NP- COMPLETENESS OF THE TASK SCHEDULING PROBLEM*

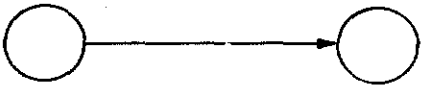
We prove in this section, the NP-completeness of the task scheduling problem on the model of Figure 4.4. It is assumed that we have identical processing orders on all machines, that is, the best permutation schedule has to be determined; and the amount of buffer space in all the machines are infinite. It is further assumed that no preemption is allowed in the schedule. We only prove for the special case of two machines on the second stage (i.e. m=2).

THEOREM 4.1

The problem of deterministic task scheduling on the SIMD model with the following assumptions, is NP-complete:

(1) m=2 (two machines on the second stage);

(2) Each request has the following task precedence graph:



$0<P_i(M_a)<\infty$ $\qquad$ $0<P_i(M_{b,j})<\infty$

$j=\{1,2\},\ i\in\{1,\ldots,N\}$

That is, each request only requires the service of one machine on the second stage. There are no precedence constraints among requests;

(3) The optimization criterion is to minimize $C_{\max}$, the maximum task completion time.

Proof

Problem ∈ NP because a non-deterministic Turing Machine can predict the sequence in polynomial time.

The problem is reducible from the knapsack problem[1].

Let n=t+2

$$\forall i \in T \quad P_i(M_{b,2})=0; \quad \sum_{i\in T} P_i(M_a)=A; \quad \sum_{i\in T} P_i(M_{b,1})=B;$$

Jobs are agreeable if

$$\sum_{S\subset T} P_i(M_a) \overset{=}{<} a \ then \ \sum_{S\subset T} P_i(M_{b,1}) \overset{\geq}{<} b$$

where $P_i(M_a),\ P_i(M_{b,1}) \gg 0$

$$P_{n-1}(M_a)=1; \quad P_{n-1}(M_{b,1})=a; \quad P_{n-1}(M_{b,2})=0;$$

$$P_n(M_a)=b-A+a; \quad P_n(M_{b,1})=0; \quad P_n(M_{b,2})=B-b+A-a;$$

$$y = B+a+1$$

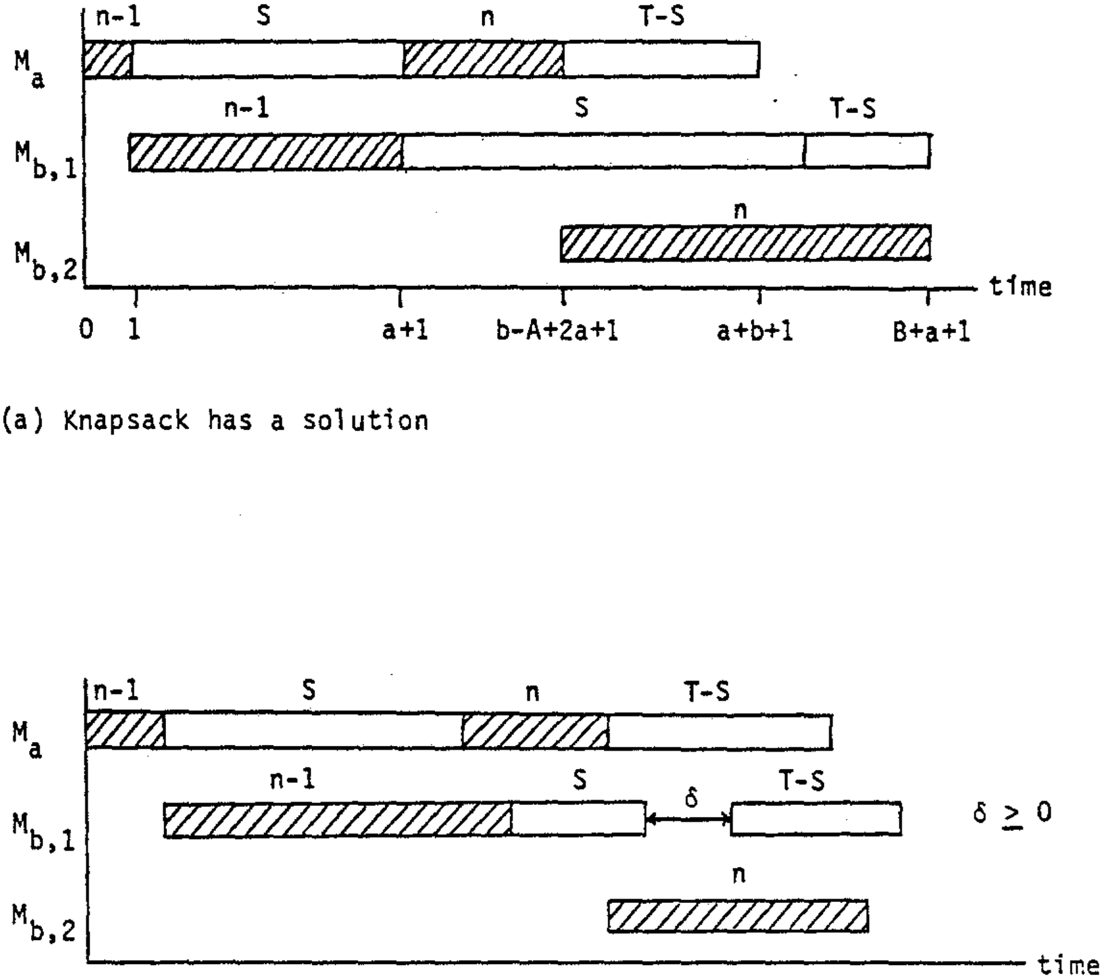
If knapsack has a solution, then there exists a schedule with $\sum_{i\in S} P_i(M_a)=a$ and $C_{\max}=y$ as illustrated in Figure 4.6a. If Knapsack has no solution, then $\sum_{i\in S} P_i(M_a)-a=c\neq 0$ for each $S\subset T$ and we have a processing order $(J_{n-1};\ \{J_i : i\in S\};\ J_n;\ \{J_i : i\in T-S\})$ such that:

$$c>0 \ => \ C_{\max} = 1+\sum_{i\in S} P_i(M_a)+P_n(M_a)+P_n(M_{b,2}) = B+a+c+1 > y$$

c<0 => *see Figure* 4.6*b*.

If $\sum_{i\in S} P_i(M_{b,1})<b$, then there exists overlap in between the operation of $P_j(M_a)$ and $P_j(M_{b,1})$ for $j\in T-S$. The maximum finish time is:

[1] The knapsack problem is: "Given positive integers $a_1, \ldots, a_t$, $A=\sum_{i=1}^{t} a_i$, B, does there exist a subset $S\subset T=\{1,\ldots,t\}$ such that $\sum_{i\in S} a_i=B$.



(a) Knapsack has a solution

(b) Knapsack has no solution and $c < 0$

Figure 4.6 Proof of Theorem 4.1

$C_{\max} = 1+a+B+\delta > y$

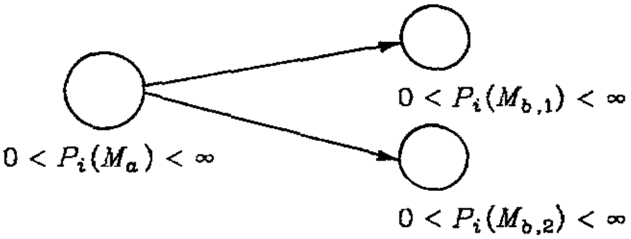
It follows that Knapsack has a solution iff this problem has a solution with $C_{\max} \leqq y$. Since the Knapsack problem is NP-complete, the problem we are considering is also NP-complete.

Q.E.D.

THEOREM 4.2

The problem of deterministic task scheduling on the SIMD model with assumptions similar to Theorem 4.1 except for assumption (2), is NP-complete.

(2) Each request has the following task precedence graph:



That is, each request requires the service of both machines on the second stage. There are no precedence constraints among requests.

Proof

Problem ∈ NP because a non-deterministic Turing Machine can predict the sequence in polynomial time.

The problem is reducible from the knapsack problem.

Let

n=t+1

$P_i(M_a)=1;\ \ P_i(M_{b,1})=t*a_i;\ \ P_i(M_{b,2})=1;\ \ (i \in T);$

$P_n(M_a)=t*b;\ \ P_n(M_{b,1})=1;\ \ P_n(M_{b,2})=t(A-b)+1;$

$y=t(A+1)+1;$

The timing diagram is shown in Figure 4.7.

If knapsack has a solution, the ∃schedule with $\sum_{i \in S} P_i(M_a)=b$ and $C_{\max}=y$.

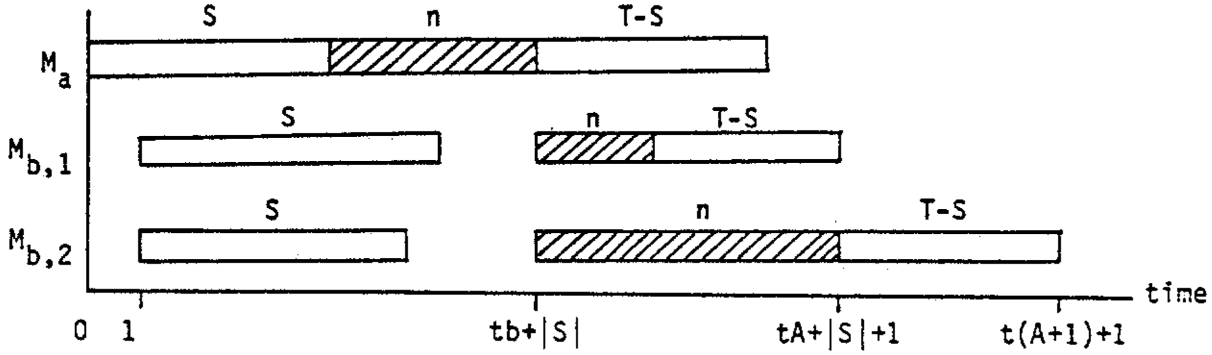


Figure 4.7 Timing Diagram for the Proof of Theorem 4.2

If knapsack has no solution, then $\sum_{i\in S} P_i(M_a) - b = c \neq 0$ for each $S \subset T$, and we have a processing order $(\{J_i : i \in S\}; J_n; \{J_i : i \in T-S\})$ such that:

$$
\begin{aligned}
c>0 \ &=> \ C_{\max} > \sum_{i\in S} P_i(M_a) + P_n(M_{b,2}) = t\,(\sum_{i\in S} P_i(M_a)) + t\,(A-b)+1 \\
&= t\,(A+c)+1 \geqq y \\
c<0 \ &=> \ C_{\max} > P_n(M_a) + P_n(M_{b,1}) + \sum_{i\in T-S} P_i(M_{b,1}) \\
&= t*b+1+t*A - t\sum_{i\in S} P_i(M_a) \\
&= t\,(A-c)+1 \geqq y
\end{aligned}
$$

It follows that Knapsack has a solution iff this problem has a solution with $C_{\max} \leqq y$. Since the Knapsack is NP-complete, the problem we are considering is also NP-complete.

Q.E.D.

THEOREM 4.3

The problem of deterministic task scheduling on the SIMD model with the following assumptions is NP-complete:

(1) m=2;

(2) There exists precedence constraints among the requests;

(3) The optimization criterion is to minimize $C_{\max}$;

Proof

Problem ∈ NP because a non-deterministic Turing Machine can guess the sequence in polynomial time.

The problem can be reduced from a conventional two stage flow shop problem with a tree precedence graph and the optimization criterion is to minimize $C_{\max}$. The reduction of the problem is obvious and will not be presented here. Since the two stage flow shop problem with a tree precedence graph is NP-complete, this implies that the problem we are considering is NP-complete as well.

Q.E.D.

THEOREM 4.4

The problem of deterministic task scheduling on the SIMD model with the following assumptions is NP-complete:

(1) m=2;

(2) The release dates of jobs may be $\geqq 0$, that is, not all jobs are available initially;

(3) The optimization criterion is to minimize $C_{\max}$;

Proof

Problem $\in$ NP because a non-deterministic Turing Machine can guess the sequence in polynomial time.

The problem can be reduced from a conventional two stage flow shop problem with release dates $\geqq 0$ and the optimization criterion is to minimize $C_{\max}$. The reduction is obvious. Since the two stage flow shop problem with positive release dates is NP-complete, this implies that the problem we are considering is NP-complete.

Q.E.D.

THEOREM 4.5

The problem of deterministic task scheduling on the SIMD model with the following assumptions is NP-complete:

(1) m=2;

(2) There are no buffers on the second stage, i.e. $buff\,(M_{b,1})=buff\,(M_{b,2})=0$. There will be no waiting of requests on the second stage;

(3) The optimization criterion is to minimize $C_{\max}$.

Proof

Problem ∈ NP because a non-deterministic Turing Machine can guess the sequence in polynomial time.

The problem is reducible from the knapsack or the partition problem.

Let:

$$n=t+2;$$

$$\forall i \in T;\quad P_i(M_a)=a_i;\quad P_i(M_{b,1})=0;\quad P_i(M_{b,2})=1;\quad a_i \geqq 1;$$

If A is even, then

$$P_{n-1}(M_a)=1;\quad P_{n-1}(M_{b,1})=\frac{A}{2}+1;\quad P_{n-1}(M_{b,2})=0;$$

$$P_n(M_a)=1;\quad P_n(M_{b,1})=\frac{A}{2}+1;\quad P_n(M_{b,2})=0;$$

$$y=A+3;$$

If A is odd, then

$$P_{n-1}(M_a)=2;\quad P_{n-1}(M_{b,1})=\frac{A+3}{2};\quad P_{n-1}(M_{b,2})=0;$$

$$P_n(M_a)=2;\quad P_n(M_{b,1})=\frac{A+3}{2};\quad P_n(M_{b,2})=0;$$

$$y=A+5.$$

If knapsack (for A odd) or partition (for A even) has a solution, then there exists a schedule with $\sum_{i \in S} P_i(M_a)=\frac{A}{2}$ and $C_{\max}=y$ as illustrated in Figure 4.8.

If knapsack (partition) has no solution, then $\sum_{i \in S} a_i-\frac{A}{2}=c\neq 0$ for each $S \subset T$ and we have a processing order $(J_{n-1};\ \{J_i{:}i \in S\};\ J_n;\ \{J_i{:}i \in T-S\})$ such that for A even,

$$\begin{aligned} c>0 \;=>\; C_{\max} &= P_{n-1}(M_a)+\sum_{i \in S} P_i(M_a)+P_n(M_a)+P_n(M_{b,1}) \\ &= 1+\frac{A}{2}+c+1+\frac{A}{2}+1 \\ &> y \end{aligned}$$

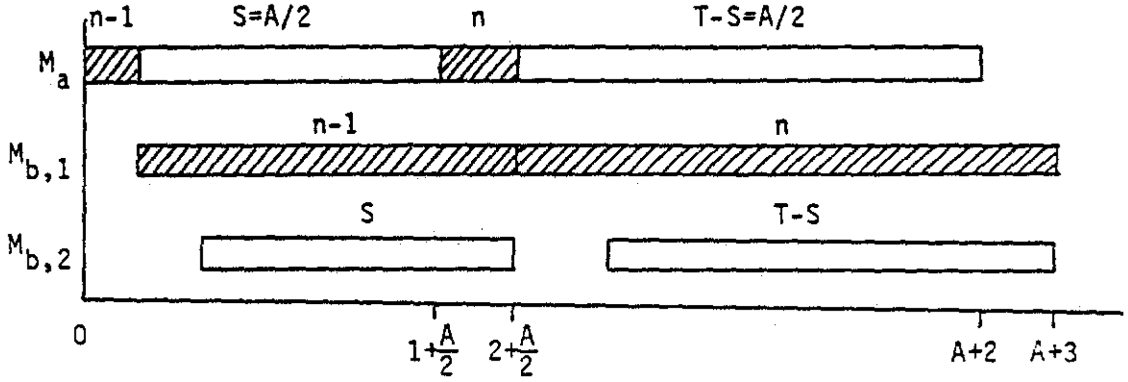


Figure 4.8 Timing Diagram for the Proof of Theorem 4.5 (A even)

$$
\begin{aligned}
c<0 \;=>\; C_{\max} &= P_{n-1}(M_a)+P_{n-1}(M_{b,1})+\sum_{i\in T-S} P_i(M_a)+1 \\
&= 1+\frac{A}{2}+1+\frac{A}{2}+c+1 \\
&> y
\end{aligned}
$$

Note that in this case, although the jobs in S have finished, job n cannot be started until time=$1+\frac{A}{2}$ because there is no buffer available in $M_{b,1}$.

It follows that knapsack has a solution iff this problem has a solution with $C_{\max} \leq y$. Since the knapsack problem is NP-complete, the problem we are considering is also NP-complete.

Q.E.D.

We have therefore proved that the task scheduling problem on the SIMD model is NP-complete (with the assumptions stated in the theorems). An approach we can take now is to design a suitable heuristic for each of these problems. However, we delay this until Section 4.5. In the next section, we show by restricting the processing time on each machine that the task scheduling problem can be made polynomially solvable. The processing times of the tasks are restricted in a fashion such that $P_i(M_a)=1$ and $P_i(M_{b,j})=m$ (m=number of machines on the second stage) and each request needs the service of only one machine on the second stage. This particular model represents a model of an interleaved memory system.

4.4 *THE RESTRICTED MODEL - AN OPTIMAL ALGORITHM FOR SCHEDULING REQUESTS ON AN INTERLEAVED MEMORY SYSTEM*

4.4.1 *Requirements for the Design of a Primary Memory*

In a top-down design, the requirements and the attributes must first be identified before the system can be designed. Requirements are the constraints which the system must satisfy and they reflect the environment as well as the objectives of the system. Attributes, on the other hand, specify either options or evaluation criteria for qualitative comparisons of competitive systems that meet the system requirements. Attributes may be used to evaluate the tradeoffs in competing architectures and to obtain a feeling for the 'goodness' of the architecture in realizing the system. The requirements for the design of a primary memory are:

(1) *Bandwidth*

The bandwidth represents the average throughput of the memory system and is given in terms of bits returned/unit time. In a parallel memory system, the bandwidth is the sum of the bandwidths of all the modules (Bandwidth = $\sum_{module\ k}$ (word length of module k)*(average utilization of module k)/(cycle time of module k) where the average utilization of a module is the average fraction of time the module is busy. For the case of identical modules, the bandwidth can be written as:

$$Bandwidth = \frac{\begin{bmatrix} number\ of \\ modules \end{bmatrix} * \begin{bmatrix} word \\ length \end{bmatrix} * \begin{bmatrix} average \\ utilization \end{bmatrix}}{(speed\ of\ module)}$$

$$Bandwidth = \frac{constant * \begin{bmatrix} average\ number\ of \\ busy\ modules \end{bmatrix}}{(memory\ cycle\ time)} \qquad (4.1)$$

where the constant in Eq. 4.1 has a unit of (bits * memory cycle / unit time). The model of interleaved memories presented here assumes that all the modules are identical and the word length of each module are kept constant. The objective of maximizing the bandwidth is therefore equivalent to maximizing the average utilization of the modules.

(2) *Response time*

The response time is the delay between the time a request is accepted by the primary memory and the time the request is serviced, assuming that the datum resides in the primary memory. This is also called the waiting time of the requests.

(3) *Size*

This is the required memory size or capacity.

(4) *Cost*

This is the maximum allowable cost of the resultant design which satisfies the above requirements.

The design of the memory must satisfy the above requirements. Moreover, the performance of the final system can be evaluated by using these parameters as evaluation criteria.

4.4.2 *Characteristics of the Access Sequence of a Pipelined Processor*

In this section, we describe the characteristics of the access sequence of a pipelined processor. A pipelined organization in the most general sense, instead of specially structured pipelined computers with different arithmetic units (e.g. CRAY I), applications (e.g. vector processing), additional memory support (e.g. cache) and interconnections (e.g. ILLIAC IV), is assumed. The processor is further assumed to be executing directly from the main memory. The scheduling algorithms developed are general enough to be applicable to the interleaved memories of all the specially structured pipelined computers. However, the exact performance is not found for each type of machine.

A memory access sequence generated by a pipelined processor has Class D dependencies as classified by Chang et. al. [CHA77]. A dependency is a logical relationship between two addresses such that the second address cannot be accessed (written or read) until the first has been accessed. Class D dependency is characterized by a machine with instruction level multiprogramming (from a large number of jobs), or a machine with sufficient lookahead or queueing hardware to allow dependencies to be bypassed. However, there still exist cases where the effects of dependencies cannot be eliminated. Anderson et. al.

have identified three main sources of concurrency limitations which tend to reduce the performance of the pipe [AND67]. These are:

(a) Register interlock - When the current instruction needs a register modified by a previous instruction, the current instruction cannot be decoded until the previous instruction has finished;

(b) Branching - When a jump or a branch on condition instruction is encountered, further operations in the pipe cease until the target instruction has returned from the memory. Conditional branching poses an additional delay because the branch decision depends on the outcome of arithmetic operations in the execution units.

(c) Interrupts - When an interrupt occurs in the pipe, it is necessary to sequentialize the execution of instructions in the pipe in order to determine the exact source of the interrupt. This sequentialism in execution would degrade the performance of the pipe.

Various methods have been introduced to solve these dependency problems [TOM67]. For example, regiester interlocks can be solved by using forwarding; the sequentialism due to interrupts can be eliminated by using imprecise interrupts as in IBM 360/91. The most predominant effect on the performance of the memory is due to branching. When a branch or a conditional branch instruction is encountered, request supply to the memory discontinues until the condition code has been set and the target instruction has returned from the memory. The utilization of the memory therefore decreases. The effects on the memory performance due to branching dependencies are studied in section 4.4.9.

In addition to the effects due to address dependencies, the order in which instructions and data are requested also affects the memory performance. For a pipelined processor, the request stream is a sequence of instruction-operand fetch pairs. However, not every instruction involves an operand fetch and if the bus is wide enough, two or more instructions can be fetched in one access. A notable characteristic in this access pattern is that instruction fetches are made in a sequence interlaced with operand accesses. The performance of the memory system may be improved by separating the memory modules into two sets, one for instructions and one for data. In section 4.4.8, the effects on memory performance due to separation and mergence of instruction and data modules are compared.

4.4.3 *Previous Work on the Study of Interleaved Memories*

One of the early successful implementation of interleaved memories is in the IBM 360/91 [BOL67]. In this computer, the storage system is made up of an interleaved set of memory modules and the degree of interleaving equals the number of memory modules. The memory can service a string of sequential requests by starting, or selecting, a storage unit every cycle until all are busy. In effect, the storage cycles are all staggered (see Fig. 4.10). By using a set of buffers called the request stack, conflicting requests which access the same module can be resolved by allowing only one of these requests to access the module and storing the rest in the request stack to be issued in later cycles. Simulation results were shown for the average access time and the bandwidth with various degree of interleaving.

The earliest attempt to model the performance of interleaved memories was done by Hellerman [HEL67]. By assuming a saturated request queue (a queue in which requests are never exhausted) with random requests, and no provision is made for the queueing of the requests on busy modules, the request queue is scanned until a repeated request is found. This constitutes a collision. Hellerman's results show that with m memory modules, the average number of requests scanned before a collision is approximately $m^{0.56}$ for m between 1 and 45. This is taken to be an indication of bandwidth. Knuth and Rao [KNU75] show an alternate exact way to calculate the bandwidth. However, both of these results are pessimistic because they do not allow the queueing of conflicting requests to the same module and the randomness assumption is not tenable in real programs.

Burnett et. al. have developed a number of models on parallel memories. In two of these models, [BUR70, BUR73], they assume that the modules operate synchronously (all modules start and end their cycles simultaneously) and a scanner scans a saturated request queue and admits new requests to service until it attempts to assign a request to a busy module. In two other models, [COF71, BUR75], they further assume that a set of blockage buffers is present so that requests made to a busy module can be stored and issuued in later cycles. The scanner continues to scan the request queue until all the modules have been allocated or all the buffers are occupied. In effect, the maximum size of the request queue inspected by the scanner never exceeds b+m where b is the number of buffers and m is the number of memory modules. They have also studied a request model similar to Strecker's model [STR70] by assuming a probability α for the succeeding request to request the next module in sequence and a probability of $(1-\alpha)/(m-1)$ to request any other module. They have developed two algorithms that modified the request pattern in order to increase the bandwidth. The first one is called the Instruction-Data Cycle Structure, which distinguish the request queues into two sub-queues, the instruction queue and the data queue. These two sub-queues are inspected in alternate memory cycles. They found that there are improvements from -4% to 12% in bandwidth (the number of modules varies from 8 to 16) over a model with four blockage buffers and a single queue [BUR75]. The second algorithm, the Group Request Structure, separates a memory cycle into two sub-cycles, the first sub-cycle is used for servicing the instruction queue, and the second sub-cycle is used for servicing the data queue. They found that there are 8% to 16% improvements over the same Instruction-Data Cycle Structure algorithm. Terman [TER76] has made a trace driven simulation on the Instruction-Data Cycle Structure algorithm and found that the theoretical predictions of Burnett and Coffman fit well with the simulation results for the fetching of instructions, but their predictions do not fit well with the simulation results for data requests which are more random than instruction requests and are difficult to be modelled accurately.

Many other researchers have studied models of parallel memories. These include Flores [FLO64], Skinner and Asher [SKI69], Ravi [RAV72], Bhandarkar [BHA75], Sastry and Kain [SAS75], Baskett and Smith [BAS76], Briggs and Davidson [BRI77], Chang, Kuck and Lawrie [CHA77], Smith [SMI77] and Hoogendoorn [HOO77]. These studies are directed toward multi-processor systems and we will not describe them here.

In the remainder of this section, the deficiencies found in the previous models are summarized.

(1) All the previous models assume that the memories operate synchronously. As Burnett and Coffman pointed out, simultaneous memory operations offer more opportunity to take advantage of program behavior in a particular memory system [BUR75]. However, with synchronous operations, there is the problem of returning the results of the accesses from the memory. Since the results from each module are available simultaneously, extra data paths or queues are needed to return these data to the processor. Further, a pipelined processor usually makes requests in sequence rather than in batches. Therefore it is desirable to study a model in which the memory modules operate out of phase. By out of phase, we mean either a) the initiations of the modules are asynchronous or b) the initiations of the modules are timed by a clock and during a clock interval, at most one module can be initiated.

Because the operations of asynchronous modules are much more difficult to control, only case (b) is considered in this design.

(2) Very few studies have been made to minimize the waiting time of a request to the memory. Flores [FLO64] has made a quantitative study relating the waiting time factor to the memory cycle time, the input/output time and the worst case execution time for different numbers of memory banks. However, his studies were directed toward the effect of interference from the input/output units and there was no queueing of requests. In other models, a saturated request queue is assumed, and the effects of waiting time are not considered. When the queue size is finite, it is possible to develop algorithms which optimize for the amount of waiting time in the queue, e.g. minimize the average waiting time of requests in the queue. In this section, the amount of queued requests is assumed to be finite so that the effects of waiting time can be studied.

(3) None of the previous work considers the effects of dependencies on the memory performance. Request supply to the memory ceases when a dependent instruction is executed until the dependency has been resolved. The effects of dependencies are difficult to determine because they vary strongly with the configuration of the pipe and the strategies employed in the pipe to resolve them. Request rate to the memory may also decrease for other reasons. For example, in the IBM 360/91, there is a small amount of instruction buffers in the CPU which serve as another level of the memory hierarchy. When a small loop occurs such that all the instructions of the loop fit in the instruction buffers, instruction accesses to the memory stop until execution of the loop is finished. Other machines may have different approaches. However, the evaluation of memory performance for a specific machine is too restrictive. We

take an approach which first evaluates the performance for the general case of an interleaved memory with a saturated, non-dependent request stream. The degradation in performance due to dependencies in the requests is then estimated subsequently.

4.4.4 *The Organizations of Primary Memory for a Pipelined Processor*

We present in this section two different implementation alternatives of interleaved memories (Organization I and Organization II). The two organizations differ in the configurations of the request buffers. In Organization I, a single set of request buffers is assumed to be shared by all the modules and in Organization II, individual request buffers exist for each module. The general assumptions made are as follows:

(1) The request rate from the processor is assumed to be high enough so that any empty buffer in the memory system is filled up by an incoming request immediately. Buffers are assumed to exist at the processor end so that any additional requests generated by the processor can be queued there. The requests that can be served by the modules are those that exist in the buffers only. This assumption is made because we want to get an upper bound on the performance of the memory. In a practical system, the memory is usually the bottleneck and our assumption is therefore valid.

(2) Each request is assumed to be an integer from 0 to m-1, which is the module it requests, and is obtained as the residue of dividing the address by m.

(3) The service time of each module (the read time or the write time) for a request is assumed to be constant. This is a good model for semiconductor memories. We also assume that a memory module, once initiated to start a memory cycle, is not available until the end of the cycle.

(4) A memory cycle time is the time it takes for a memory module to service a request. Each memory cycle is assumed to consist of m equally spaced memory sub-cycles. It is further assumed that exactly one module can be initiated to service a request at the beginning of a memory sub-cycle and it takes m sub-cycles (1 memory cycle) to service the request for all the modules i.e., homogeneous service times. With this assumption, the problem of multiple data paths is resolved because at most one module finishes in each sub-cycle and the system is never confronted with returning results from more than one module simultaneously. The modules are therefore clocked by the memory sub-cycles.

In Organization I (Fig. 4.9), there are m memory modules; a single set of b+1 associative buffers, B_T, B_1, B_2, ..., B_b; and an intelligent scheduler which schedules a memory module to start a memory cycle. The modules operate out of phase in a fashion called staggered cycles. One example of a staggered cycle is shown in Fig. 4.10. The set of b+1 associative buffers are used to store incoming requests. A request queued on a specific module can be retrieved in one associative search operation. Whenever a request is taken out from a buffer, all the requests behind it are pushed one location up so that B_T is empty. The buffer B_T has an additional function, namely, to receive requests from the bus. Due to our assumption of high request rate, B_T is filled immediately whenever it is empty. The queueing discipline for the requests in the buffers directed towards the same module is essentially First-In-First-Out (FIFO). Other queueing disciplines are not studied because only uni-processor systems are considered in this design.

The center of the control in the memory system is the intelligent scheduler. The scheduler, using a scheduling algorithm, decides at the beginning of each memory sub-cycle whether to initiate a memory module and if so which module to initiate. The selection of which module to initiate is determined by the

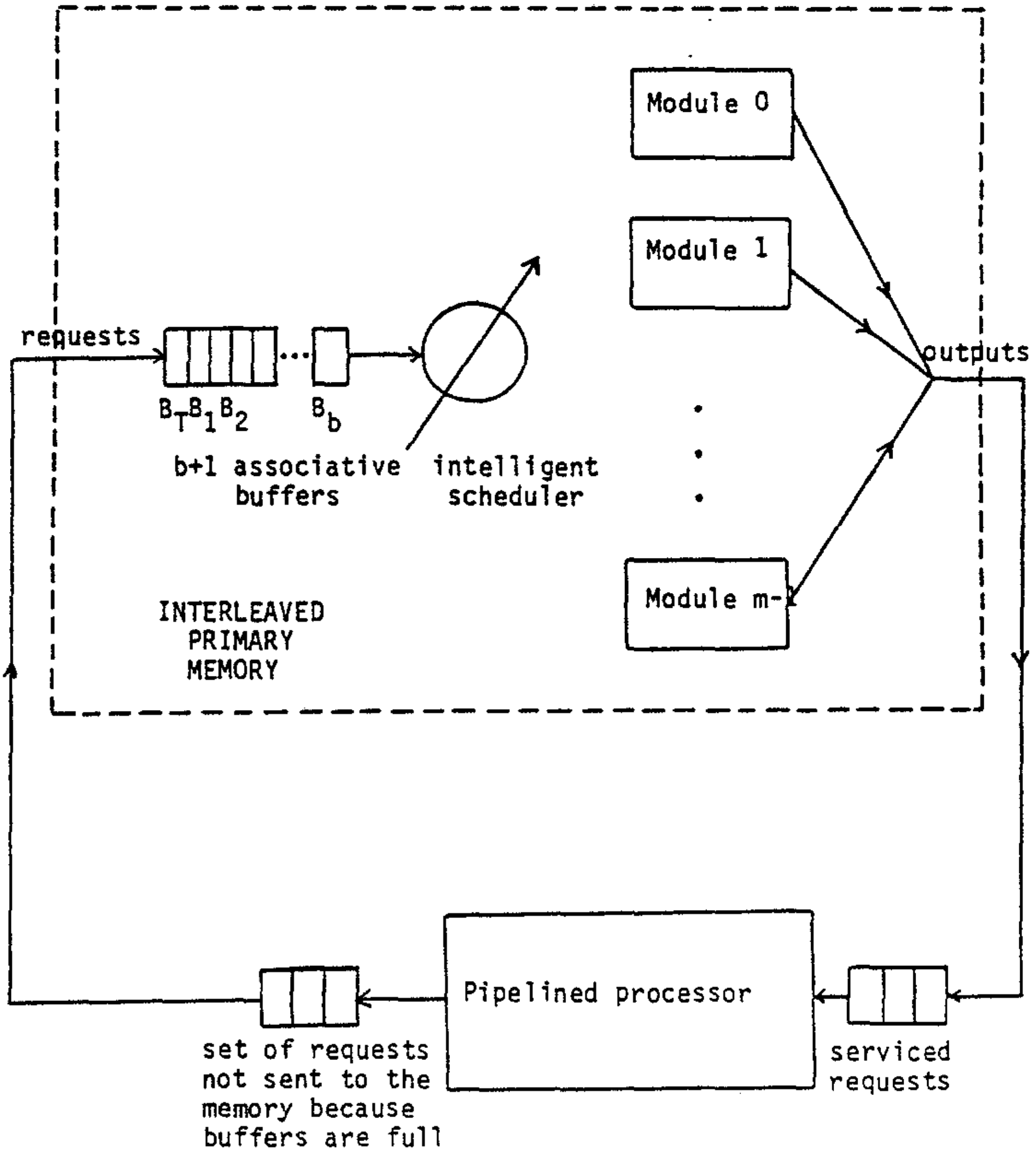


Figure 4.9 Org. I - A Model of Interleaved Memories with a Single Request Queue

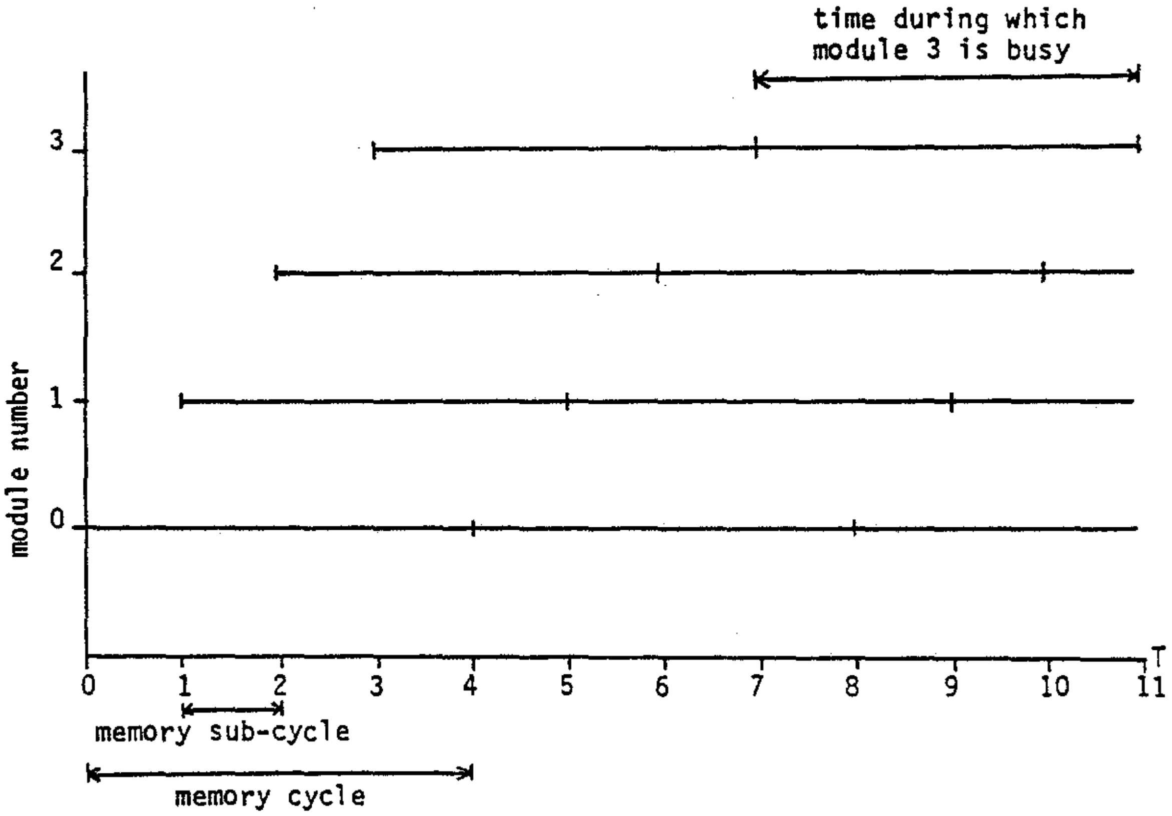


Figure 4.10 A Gantt Chart to illustrate the Operations of the Interleaved Memories in Staggered Cycles (m=4)

information about the requests in the associative buffers and by the knowledge about the status of the modules (free or busy). Three scheduling algorithms are investigated in this design.

(1) *Algorithm 4.1 Round- Robin (RR)*

All the modules are initiated in a round-robin fashion regardless of whether a request is queued on the module. The scheduler does not make use of any information about the status of the system. The implementation of this algorithm is very simple and the scheduler only has to konw the current module initiated. In Fig. 4.10, the Gantt Chart for the operation of a 4-way interleaved memory with RR scheduling algorithm is shown. This is the scheduling algorithm that is implemented in most interleaved memory systems today.

(2) *Algorithm 4.2 First- Free- First (FFF)*

In this âlgorithm, only the information about the status of the modules (free or busy) is utilized by the scheduler. There is a FIFO list of free modules. At the beginning of a memory sub-cycle, the scheduler puts a busy module to the end of the free list if this module finishes its cycle. It will then initiate the module at the head of the free list if there are any requests queued on it, otherwise the module at the head is appended to the tail of the free list and no other modules are checked in this cycle. The scheduler may also check all the subsequent modules in the free list, but the time for this is proportional to the number of modules and is not feasible when this number is large.

(3) *Algorithm 4.3 Maximum- Work- Free- Module- First (MWFMF)*

In this algorithm, both the information about the status of the modules and the requests in the buffers are utilized by the scheduler. There is a dynamic list of free modules. Conceptually, at the beginning of a memory sub-cycle, the buffers are checked associatively to see if any requests are queued on the free modules. If there is none, no module is initiated. If at least one exists, an associative search is made on the buffers and the module with the maximum number of requests queued on it is initiated. In case of ties, only the first one is initiated (Fig. 4.11a). The implementation of this algorithm can be done by using an additional associative memory of size m in the scheduler (Fig. 4.11b). Each word in this associative memory can function as a counter and is used to indicate the number of requests queued on the corresponding module. The corresponding word is incremented/decremented when a request enters/leaves the request buffers. The free module with the maximum number of requests can be obtained by performing a maximum search on those words in this associative memory corresponding to the free modules, e.g. [RAM78a] (see the associative memory design in Chapter 5). The maximum search algorithm shown in [RAM78a] is parallel by word and serial by bit and the time to perform a maximum search is proportional to the number of bits in the memory. The speed of this algorithm is therefore proportional to $\lceil \log_2 b \rceil$.

In addition to the overhead related to the execution of the scheduling algorithm, there is also the overhead of selecting the request from the associative buffers and sending it to the memory module. This overhead consists of matching the selected module number against all the requests in the buffers and selecting the first request if multiple responses occur in the match. Using a bit-serial word-parallel equality matching algorithm, e.g. [RAM78a], and a binary tree type multiple match resolution circuit, e.g. [FOS68], this overhead is proportional to $\lceil \log_2 m \rceil$. In general, the overheads associated with the three scheduling algorithms are very small, and the selection of a module and the corresponding request to be initiated in the next sub-cycle can be overlapped

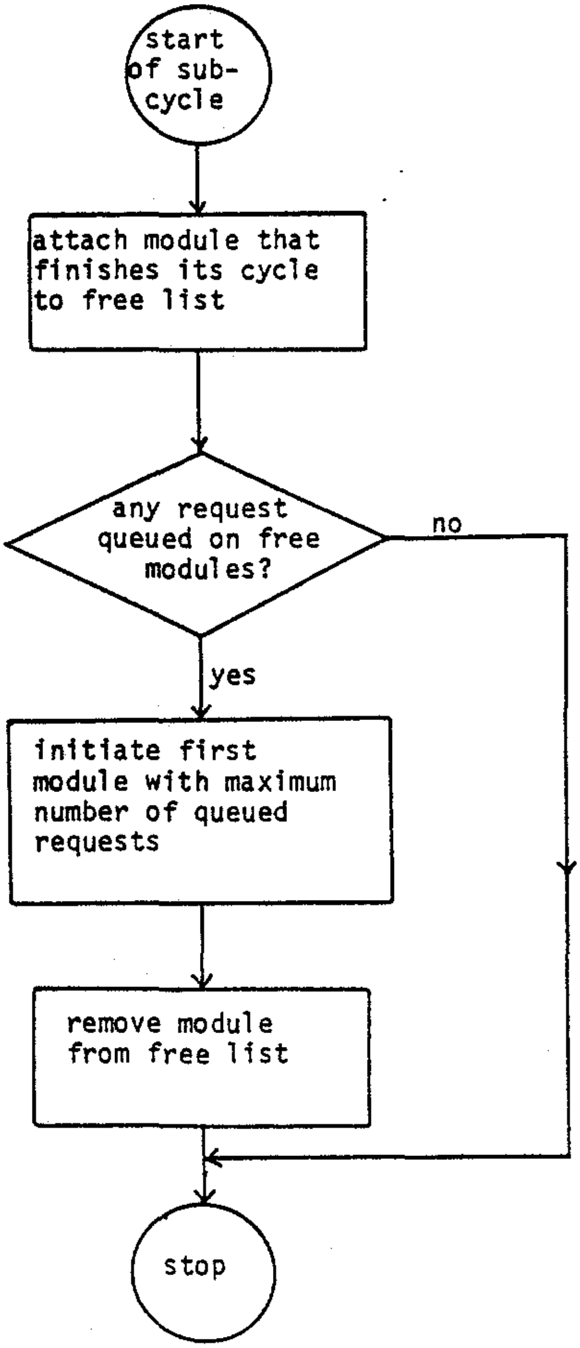

start of sub-cycle
attach module that finishes its cycle to free list
any request queued on free modules?
no
yes
initiate first module with maximum number of queued requests
remove module from free list
stop

Figure 4.11a Algorithm 4.3 - MWFMF Scheduling Algorithm

pointer to modules	busy-free status	amount of queued requests
0	0	
1	1	
2	1	
		ASSOCIATIVE MEMORY
.	.	
.	.	
.	.	
m-1	1	
$\lceil \log_2 m \rceil$	1	$\lceil \log_2 b \rceil$

Figure 4.11b Implementation of the MWFMF Scheduling Algorithm using Associative Memory

with the current sub-cycle.

At the end of each memory sub-cycle, at most one request is serviced. The result is sent back to the processor. The necessary queue for storing these results is excluded from the memory model.

The requests of the system come into the memory in a specific pattern. Two types of access patterns are considered in this design:

(1) *Random accesses with no address dependency* - All the addresses have no correlation and are independent of each other. This can be used to model the request stream from computer systems with instruction level multiprogramming or multi-processor systems where the number of processors is larger than the number of modules.

(2) *Accesses from the execution trace of a monoprogrammed pipelined computer* - The addresses in the execution traces are correlated and they represent a similar addressing behavior when the actual program is executed on a pipelined processor. We have used execution traces from a pipelined processor, representing large scientific applications, the CDC 7600, in this study.

Organization II is similar to Organization I except that separate sets of buffers exist for each module (Fig. 4.12). Requests from the processor are continuously moved into the buffers of each module via B_T until a request in B_T is directed toward a module whose buffers are already full. The request in B_T is blocked, and as a result, further requests are blocked from entering the memory. When the module responsible for this blocking has finished servicing its current request, one request from its buffers is serviced which results in an empty buffer. The blocking request in B_T is moved into this empty buffer. Because of the independent queues, one or more requests can then be accepted to the memory system until the previous blocking situation occurs with one of

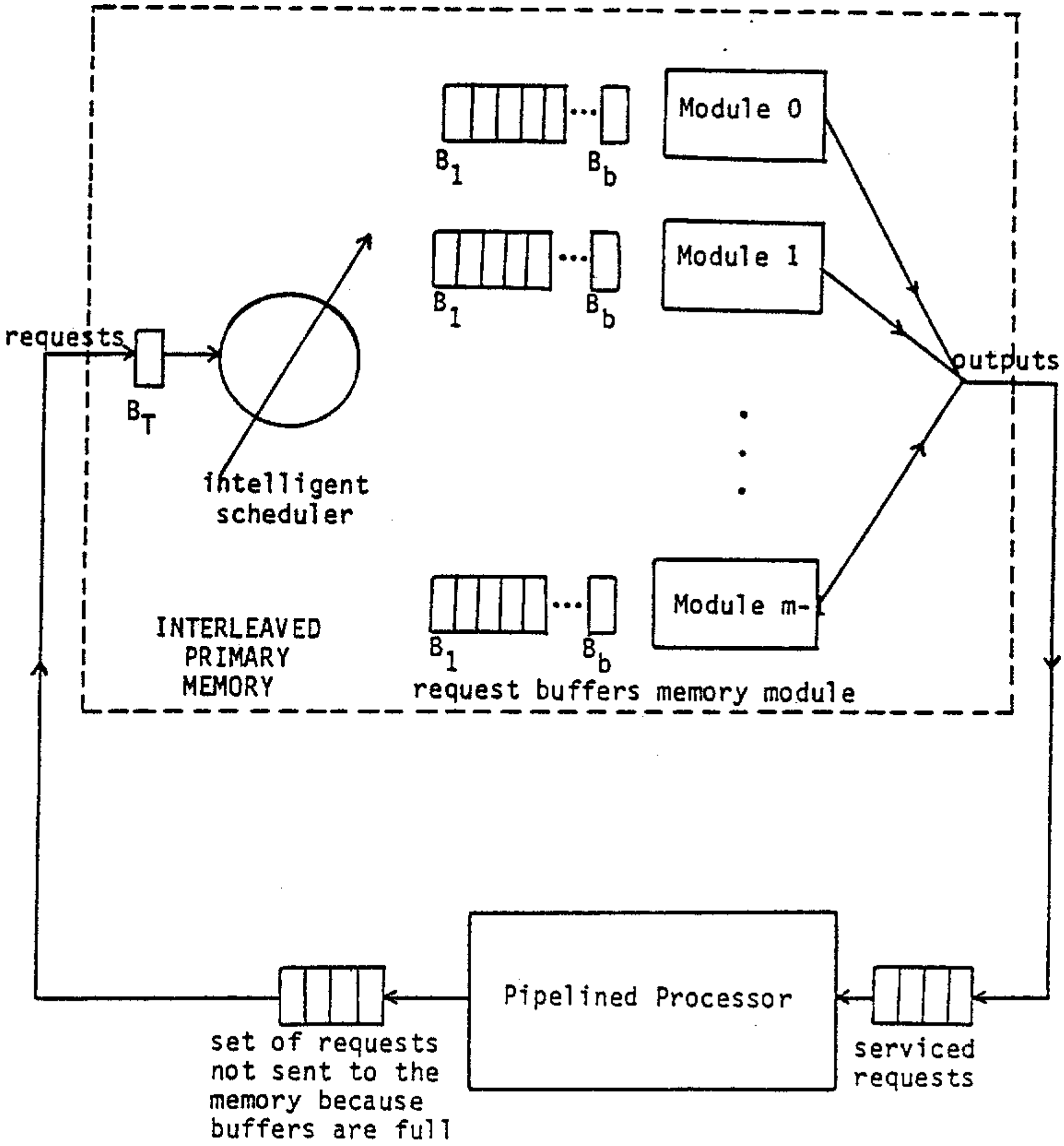


Figure 4.12 Org. II - A Model of Interleaved Memories with Multiple Request Queues

the modules. When b=0, there is only one buffer, B_T, in the system and this is exactly the same as Organization I with b=0. Therefore Organization II degenerates into Organization I when b=0. The buffers used in this organization is simpler than that of Organization I. Associative search capabilities are not necessary for these buffers. The implementation of the scheduler is similar to that of Organization I. The advantage with this system is that the request buffers are simple shift registers and therefore are cheaper. However, in order for this organization to operate at full capacity, more than one request may have to be moved across the bus into the memory in a memory sub-cycle. As we recall, we assume that a pipelined processor generates in the order of one request every memory sub-cycle, therefore, the blocking situation may not always occur and the buffers are under-utilized. Further, it is necessary to build a faster bus so that multiple requests can be moved across the bus in a memory sub-cycle. We can assume that sufficient requests are queued in the processor so that the need of moving more than one request into the memory system during a sub-cycle can be satisfied. An alternative is to allow a maximum of one request to be accepted in every sub-cycle. This results in a degraded performance for Organization II because the system is not operating with the maximum request rate.

Since the two organizations discussed are operating in steady state and the systems discussed are balanced, the average arrival rate and the average waiting time are related by Little's Formula.

Let

e_B = utilization of the buffers $B_1, \ldots, B_b$

(=1 for Organization I)

e_T = utilization of buffer B_T

(=1 for both organizations)

B = number of buffers in $B_1, \ldots, B_b$

(=b for Organization I; =m*b for Organization II)

$u_{m,b}$ = expected utilization of the modules

$w_{m,b}$ = expected waiting cycles of the requests

M = expected number in the system

λ = expected arrival rate

W = expected waiting time of the requests

Then

$$M = (e_B * B + 1) + u_{m,b} * m \tag{4.2}$$

$$\lambda = u_{m,b} \tag{4.3}$$

$$W = m * w_{m,b} \tag{4.4}$$

and they satisfy Little's Formula,

$$M = \lambda * W$$

Eq (4.3) is true because in a balanced system, the expected arrival rate equals the expected service rate. The physical importance of Little's Formula lies in the fact that the average utilization and the average number of waiting cycles are related. Once one of them is obtained, the other can be calculated easily. Further, it also shows that Organizations I and II are equivalent as far as the average behavior is concerned. The only difference lies in the buffer utilization which is less than 1 in Organization II whereas the buffers are fully utilized in Organization I. In the next section, we present our evaluations for Organization I only because the two organizations are equivalent and the results are directly applicable. It is shown that the MWFMF algorithm minimizes the average completion time of the requests. This result only demonstrates that the MWFMF algorithm is superior, but the exact throughput values of the system cannot be obtained analytically. The techniques that are used to evaluate the performance of these two organizations are embedded Markov analysis with random requests, and simulations with random requests and execution traces and they are shown in Sections 4.4.6 and 4.4.7.

4.4.5 *Optimality of the MWFMF Scheduling Algorithm*

In proving the optimality, it is assumed that the requests in the request queue are independent, randomly generated and of a finite size. The size of the associative buffers may be greater than, equal to, or less than the number of requests in the request queue. In a pipelined processor, memory requests can be generated continuously until a dependency occurs. At this point, the request stream is discontinued until the dependency has been resolved. Because of the high request rate assumption, the requests generated between two dependencies can be assumed to exist in the request queue after the first dependency has been resolved. However, in a practical implementation, the pipelined processor is only able to look ahead a fixed amount of instructions and this is modelled by a fixed and finite amount of associative buffers in the system (which may be greater than, less than or equal to the size of the request queue). The intelligent scheduler is allowed to examine the associative buffers in making the scheduling decision. The objective of the scheduling algorithm is to complete the service of the requests in the request queue as fast as possible so that the throughput of the memory is maximized. The symbols used in the following theorems are:

b = number of associative buffers - 1;

m = number of memory modules;

N = total number of requests that have to be serviced between two dependencies;

$\{(l_1, i_1), (l_2, i_2), \ldots, (l_m, i_m)\}_k$ = state of the memory system,

where

(l_j, i_j) = state of module j;

l_j = number of requests queued on module j in the buffers;

$$\sum_{j=1}^{m} l_j = b+1 \text{ and } l_j \geqq 0 \quad j=1, 2, \ldots, m$$

$$i_j = \begin{cases} 0 & \text{if } \textit{module } j \textit{ is free} \\ n & 0<n<m \text{ if } \textit{module } j \textit{ is busy} \end{cases}$$

In the case that module j is busy, n is the number of cycles that module j has serviced its current request. The number of cycles remaining before the completion of service for the current request is (m-n) mod m.

k = variable used in the induction proof indicating the number of remaining requests to be serviced (not including those in the associative buffers);

$C_{\max}\{(l_1, i_1), (l_2, i_2), \ldots, (l_m, i_m)\}_k$ = maximum completion time for the state;

$EC_{\max}\{(l_1, i_1), (l_2, i_2), \ldots, (l_m, i_m)\}_k$ = expected maximum completion time for the state.

Before the main theorem can be stated, the following three lemmas must first be proved. Lemma 4.1 establishes the need for executing the MWFMF scheduling algorithm at the beginning of each sub-cycle. Lemma 4.2 establishes a basis for the induction proof of the main theorem and it also shows the optimality of the MWFMF algorithm when the buffer size is very large so that all the requests in the request queue reside in the buffers. Lemma 4.3 augments Lemma 4.2 by further showing that algorithm MWFMF minimizes the sum of completion times of all the requests.

LEMMA 4.1

(1) In a period of m sub-cycles, every module can be initiated at most once.

(2) At the beginning of each sub-cycle, at least one free module is available for scheduling.

Proof

(1) Obvious, because each module takes a time of m sub-cycles to service a request.

(2) Consider a time interval of m sub-cycles. Since at most one module can be scheduled in each sub-cycle, the total number of modules scheduled in m sub-cycles is less than or equal to m. At the beginning of its current sub-cycle, if a module is scheduled m sub-cycles ago, then it will finish its service at the current sub-cycle and is available for scheduling. If a module is not scheduled m sub-cycles ago, then the total number of modules scheduled in the last m sub-cycles is less than m. Therefore, at least one module is available for scheduling at the beginning of a sub-cycle.

Q.E.D.

LEMMA 4.2

If all the requests in the request queue reside in the associative buffers (that is, the buffers are large enough to accompany all these requests), then algorithm MWFMF minimizes the maximum completion time for independent, random requests in Organization I.

Proof

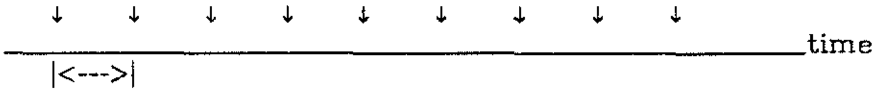
The maximum completion time is governed by the longest queue in the system. Assume without loss of generality:

$$l_1 > l_2 > \cdots > l_m$$

Case 1: $i_1 = 0$,

MWFMF schedules module 1 first.

initiate module 1



All modules will be initiated at most once in here due to lemma 4.1 (if number queued on it is non-zero) and all requests queued every module except 1 can be initiated before the last request queued on module 1 is initiated.

$C_{\max} = l_1$*m sub-cycles (initiate module 1 first)

If any other module, say module j, is initiated, then module 1 can only be initiated in the next sub-cycle after module j has been initiated.

min $C_{\max} = l_1 * m + 1$ sub-cycles (initiate module j $\neq$ 1 first)

Case 2: $i_1 > 0$

Let module j be the module such that

$i_j = 0$ and $i_1 > 0$, $i_2 > 0$, ..., $i_{j-1} > 0$.

That is, module j is the free module with the largest amount of queued requests. This will be the module scheduled by the algorithm MWFMF. In fact, the module scheduled at this point is unimportant because the maximum completion time is governed by module 1.

$C_{\max} = l_1 * m + (m - i_1)$ sub-cycles

Therefore:

min $C_{\max} = l_1 * m + (m - i_1) \bmod m$ sub-cycles

Optimum algorithm: MWFMF

On the other hand, if $l_1 = l_2 > ... > l_m$ and i_1, $i_2 = 0$, then the $C_{\max}$'s are identical whether module 1 or 2 is scheduled first. A similar proof holds for the case $l_1 \geqq l_2 \geqq \cdots \geqq l_m$.

Q.E.D.

LEMMA 4.3

If all the requests in the request queue resides in the associative buffers, then algorithm MWFMF minimizes $\sum C_j$ for independent, random requests in Organization I where C_j is the completion time for the j'th request.

Proof

Assume without loss of generality:

$$l_1 > l_2 > \cdots > l_m$$

Consider two modules a, b, such that $i_a = 0$, $i_b = 0$ and $l_a > l_b$. Let $C_{a,b}$ ($C_{b,a}$) be the sum of completion times of scheduling a before b (b before a) for modules a and b only. If b is scheduled before a, then

$$C_{b,a} = C_b + C_a = \frac{m}{2}[(l_a+1)l_a + (l_b+1)l_b] + l_a$$

Comparing this with the case of scheduling a first, it is found that:

$$C_{a,b} = C_a + C_b = \frac{m}{2}[(l_a+1)l_a + (l_b+1)l_b] + l_b$$

Since $l_a > l_b \Rightarrow C_{a,b} < C_{b,a}$, this implies that scheduling the module with a larger amount of queued requests can reduce $\sum C_j$. By adjacent pairwise interchange, it is therefore better to schedule the module with the maximum amount of queued requests if it is free. If the module is not available, scheduling the free module with the maximum amount of queued requests is also optimum.

Q.E.D.

From the proofs of Lemmas 4.2 and 4.3, it is seen by using the MWFMF algorithm that,

(1) The throughput of the memory is at a maximum because the maximum time to complete a set of jobs is minimized (Lemma 4.2).

(2) The average waiting time is minimized. This is because C_j, the completion time for the j'th job equals the waiting time for the j'th job,

$W_j = C_j - 0$, (all the jobs are available at t=0). As a result, average waiting time = $\sum W_j / M$ is also minimized (Lemma 4.3).

THEOREM 4.6

If all the requests in the request queue do not reside in the associative buffers, (that is, the buffers are not large enough to accompany all the requests in the request queue), then algorithm MWFMF minimizes the expected maximum completion time for independent, random requests in Organization I.

Proof

In order to prove this theorem, the following two parts must be proven and the theorem follows from the result of part (a).

(a) Algorithm MWFMF minimizes the expected maximum completion time for independent, random requests.

(b) Let states

$$S_1 = \{\ldots, (l_a^1, i_a), (l_b^1, i_b), \cdots\}_k$$

$$S_2 = \{\ldots, (l_a^2, i_a), (l_b^2, i_b), \cdots\}_k$$

where "..." indicates that the remaining states are identical for S_1 and S_2.

Since the states of other modules are identical, and we assume that:

$$l_a^2 > l_a^1;$$

$$l_b^1 > l_b^2;$$

$$l_a^1 + l_b^1 = l_a^2 + l_b^2;$$

and

$m \geq i_a > i_b > 0$ or $m \geq i_b > i_a > 0$ with equal probability.

If $l_a^2 > l_b^1$, then $EC_{\max}(S_1)_k \leqq EC_{\max}(S_2)_k$;

If $l_a^2 = l_b^1$, then $EC_{\max}(S_1)_k = EC_{\max}(S_2)_k$.

These two parts can be proved by induction. The truth is first established for k=0, i.e. when all the requests reside in the buffers. These parts are then assumed to be true for any positive integer k and the proof is complete by proving the case of k+1.

(I) k=0

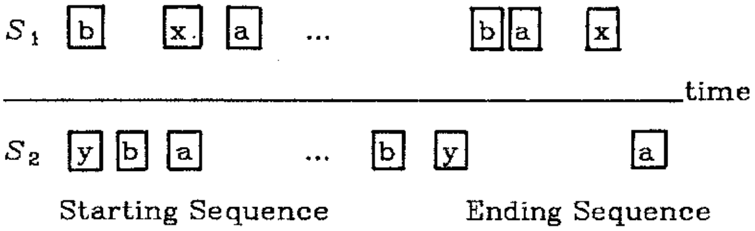
(a) MWFMF is optimal. This is established by Lemma 4.2.

(b) If there exists module z such that $l_z > l_a^2$, and since $l_a^2 \geq l_b^1 > l_b^2$ and $l_a^2 > l_a^1$, then the maximum completion time for both S_1 and S_2 depends on l_z and are identical. Therefore,

$$EC_{\max}(S_1)_0 = EC_{\max}(S_2)_0$$

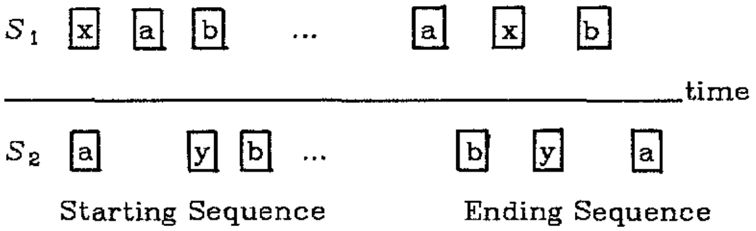
If there does not exist module z such that $l_z > l_a^2$, then the maximum completion time of S_2 depends on module a. Let there be two modules, x in S_1 and y in S_2 such that $l_a^2 > l_x^1 > l_a^1$, $l_b^1 > l_y^2 > l_b^2$ and $i_x = i_y = 0$. The following three cases can be identified.

(1) $l_a^2 > l_b^1$, $i_b = 0$, $i_a < m$



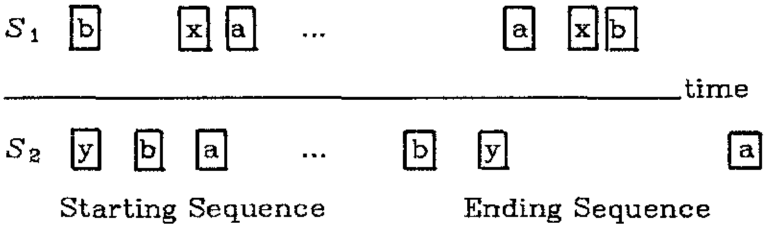
$$C_{\max}(S_1)_0 = EC_{\max}(S_1)_0 < C_{\max}(S_2)_0 = EC_{\max}(S_2)_0$$

(2) $l_a^2 > l_b^1$, $i_a = 0$, $i_b < m$

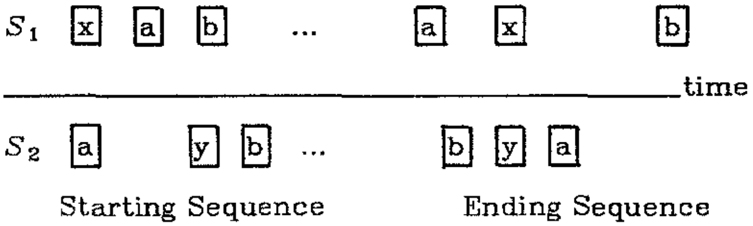


$$C_{\max}(S_1)_0 = EC_{\max}(S_1)_0 < C_{\max}(S_2)_0 = EC_{\max}(S_2)_0$$

(3) $l_a^2=l_b^1,\ i_b=0,\ i_a<m$



$l_a^2=l_b^1,\ i_a=0,\ i_b<m$



Since $l_a^2=l_b^1$, this implies that $l_a^1=l_b^2$, therefore the states S_1 and S_2 are symmetric in the states of the modules a and b and the probability that $i_b=0,\ i_a<m$ is equally likely as the probability that $i_a=0,\ i_b<m$.

$$\begin{aligned} EC_{\max}(S_1)_0 &= C_{\max}(S_1 |\ i_b=0,\ i_a<m)_0 {}^* Pr(i_b=0,\ i_a<m) \\ &\quad + C_{\max}(S_2 |\ i_a=0,\ i_b<m)_0 {}^* Pr(i_a=0,\ i_b<m) \\ &= EC_{\max}(S_2)_0 \end{aligned}$$

(II) Induction hypothesis:

Assume that the theorem is true for a positive integer k, that is,

(a) MWFMF algorithm minimizes the expected maximum completion time for independent, random requests when the number of remaining requests in the request queue is k.

(b) If $l_a^2 > l_b^1$, then $EC_{\max}(S_1)_k \leqq EC_{\max}(S_2)_k$;

If $l_a^2 = l_b^1$, then $EC_{\max}(S_1)_k = EC_{\max}(S_2)_k$.

(III) When the number of remaining inputs is k+1,

(a) Without loss of generality, let modules 1, 2, ..., j be the set of free modules. Choose any two modules, say 1 and 2, so that $l_1 > l_2$ and there does not exist p∈{1,2,...,j} such that $l_1 > l_p > l_2$. We want to compare the difference between scheduling module 1 and module 2.

(1) Schedule module 1 in this sub-cycle,

$$\{(l_1,0),\ (l_2,0),\ \ldots,\ (l_m,i_m)\}_{k+1}$$

$$=> \{(l_1-1,1),\ (l_2,0),\ \ldots,\ (l_m,(i_m+1)\ mod\ m)\}_{k+1}$$

A new input now enters the buffers, this input can be a request directed to any module in the set with equal probability 1/m (due to the assumption of independent, random requests).

New states after scheduling module 1:

1 enters: $S^1 = \{(l_1,\ 1),\ (l_2,\ 0),\ \ldots,\ (l_m,\ (i_m+1)\ mod\ m)\}_k$

2 enters: $S^2 = \{(l_1-1,\ 1),\ (l_2+1,\ 0),\ \ldots,\ (l_m,\ (i_m+1)\ mod\ m)\}_k$

- - -

m enters: $S^m = \{(l_1-1,\ 1),\ (l_2,\ 0),\ \ldots,\ (l_m+1,\ (i_m+1)\ mod\ m)\}_k$

(2) Schedule module 2 in this sub-cycle.

$$\{(l_1,0),\ (l_2,0),\ \ldots,\ (l_m,i_m)\}_{k+1}$$

$$=> \{(l_1,0),\ (l_2-1,1),\ \ldots,\ (l_m,(i_m+1)\ mod\ m)\}_{k+1}$$

New states after scheduling module 2:

1 enters: $\overline{S}^1 = \{(l_1+1, 0), (l_2-1, 1), \ldots, (l_m, (i_m+1) \; mod \; m)\}_k$

2 enters: $\overline{S}^2 = \{(l_1, 0), (l_2, 1), \ldots, (l_m, (i_m+1) \; mod \; m)\}_k$

- - -

m enters: $\overline{S}^m = \{(l_1, 0), (l_2-1, 1), \ldots, (l_m+1, (i_m+1) \; mod \; m\}_k$

It is seen that $EC_{\max}(S^1)<EC_{\max}(\overline{S}^2)$, $EC_{\max}(S^2)<EC_{\max}(\overline{S}^1)$ and $EC_{\max}(S^j)<EC_{\max}(\overline{S}^j)$ for $j \neq 1,2$. In proving $EC_{\max}(S^1)<EC_{\max}(\overline{S}^2)$, we can use the induction hypothesis II(b) and let $i_a=0$, $i_b=1$, $l_a^1=l_2$, $l_b^1=l_1$, $l_a^2=l_1$ $l_b^2=l_2$. The other parts can similarly be proved. Since the expected $C_{\max}$ is a weighted sum of the expected $C_{\max}$ of all the corresponding states, it is therefore better to schedule module 1, the module with a longer queue, first. By using the adjacent pairwise interchange argument, the free module with the maximum number of queued requests should be scheduled first.

(b) In proving this theorem, the following parts are identified.

(1) $l_a^2 > l_b^1$; Both modules a and b are not scheduled in the current sub-cycle. This can be due to (1) $i_a>0$ and $i_b>0$, i.e. both modules are busy; or (2) there exists a free module z such that l_z is greater than l_a^2 if $i_a=0$ *or* l_b^1 if $i_b=0$. Since it is assumed in the induction hypothesis II(a) that free modules with a longer queue should be scheduled, therefore module z will be scheduled in this case.

After module z is scheduled, a new input enters the buffers.

For state S_1 $\{\ldots, (l_a^1, i_a), (l_b^1, i_b), \cdots\}_{k+1}$

a enters:

$S_1^a = \{\ldots, (l_a^1+1, (i_a+1) \; mod \; m), (l_b^1, (i_b+1) \; mod \; m), \cdots\}_k$

b enters:

$S_1^b = \{\ldots, (l_a^1, (i_a+1) \; mod \; m), (l_b^1+1, (i_b+1) \; mod \; m), \cdots\}_k$

j, j $\neq$ a,b enters:

$S_1^j = \{\ldots, (l_a^1, (i_a+1) \; mod \; m), (l_b^1, (i_b+1) \; mod \; m), \cdots\}_k$

For state S_2 $\{\ldots, (l_a^2, i_a), (l_b^2, i_b), \cdots\}_{k+1}$

a enters:

$$S_2^a = \{\ldots, (l_a^2+1, (i_a+1)\ mod\ m), (l_b^2, (i_b+1)\ mod\ m), \cdots\}_k$$

b enters:

$$S_2^b = \{\ldots, (l_a^2, (i_a+1)\ mod\ m), (l_b^2+1, (i_b+1)\ mod\ m), \cdots\}_k$$

j, j $\neq$ a,b enters:

$$S_2^j = \{\ldots, (l_a^2, (i_a+1)\ mod\ m), (l_b^2, (i_b+1)\ mod\ m), \cdots\}_k$$

By the induction hypothesis,

$$EC_{\max}(S_1^a)_k < EC_{\max}(S_2^a)_k$$

$$EC_{\max}(S_1^b)_k < EC_{\max}(S_2^b)_k \qquad \text{if } l_a^2 > l_b^1+1$$

$$EC_{\max}(S_1^b)_k = EC_{\max}(S_2^b)_k \qquad \text{if } l_a^2 = l_b^1+1$$

$$EC_{\max}(S_1^j)_k < EC_{\max}(S_2^j)_k \qquad \forall j \neq a,b$$

Therefore

$$EC_{\max}(S_1)_{k+1} \leqq EC_{\max}(S_2)_{k+1}$$

(2) $l_a^2 > l_b^1$ and there exists a module x such that $i_x = 0$ and

$$l_a^2 > l_x > l_a^1 \quad \text{if } i_a=0 \text{ or}$$

$$l_b^1 > l_x > l_b^2 \quad \text{if } i_b=0$$

Let us look at the first case:

$$S_1 = \{\ldots, (l_a^1,0), (l_b^1,i_b), \ldots, (l_x,0), \cdots\}_{k+1}$$

$$S_2 = \{\ldots, (l_a^2,0), (l_b^2,i_b), \ldots, (l_x,0), \cdots\}_{k+1}$$

According to the MWFMF algorithm, module x should be scheduled in S_1 and module a should be scheduled in S_2. It is necessary to compare the expected $C_{\max}$ after these have been scheduled. Suppose module x is not scheduled in both states, from part III(b)(1), it is seen that $EC_{\max}(S_1 |\ a\ scheduled)_k < EC_{\max}(S_2 |\ a\ scheduled)_k$. However, due to the induction hypothesis, II(a), scheduling x in state S_1 would be better than scheduling a because $l_x > l_a^1$.

$$EC_{\max}(S_1 |\ x\ scheduled)_k < EC_{\max}(S_1 |\ a\ scheduled)_k$$

Therefore:

$$EC_{\max}(S_1 |\ x\ scheduled)_k < EC_{\max}(S_2 |\ a\ scheduled)_k$$

and

$$EC_{\max}(S_1)_{k+1} < EC_{\max}(S_2)_{k+1}.$$

The other case, i.e. $l_b^1 > l_x > l_b^2$ and $i_b = 0$ can be similarly proved. For the remainder of the proof of this theorem, it is assumed that $l_a^2 \geqq l_b^1 > l_x$, for all x $\neq$ a,b and $i_x = 0$.

(3) $l_a^2 > l_b^1, \quad 0 < i_a < m, \quad i_b = 0$

Due to the induction hypothesis, module b should be scheduled in S_1 and S_2.

For state S_1, schedule module b in this sub-cycle,

$$\{\ldots, (l_a^1, i_a), (l_b^1, 0), \cdots\}_{k+1}$$

$$=> \{\ldots, l_a^1, (i_a+1)\ mod\ m), (l_b^1-1, 1), \cdots\}_{k+1}$$

New input enters the buffer:

a enters: $S_1^a = \{\ldots, (l_a^1+1, (i_a+1)\ mod\ m), (l_b^1-1, 1), \cdots\}_k$

b enters: $S_1^b = \{\ldots, (l_a^1, (i_a+1)\ mod\ m), (l_b^1, 1), \cdots\}_k$

j, j $\neq$ a,b enters: $S_1^j = \{\ldots, (l_a^1, (i_a+1)\ mod\ m), (l_b^1-1, 1), \cdots\}_k$

For state S_2, schedule module b in this sub-cycle:

$$\{\ldots, (l_a^2, i_a), (l_b^2, 0), \cdots\}_{k+1}$$

$$=> \{\ldots, (l_a^2, (i_a+1)\ mod\ m), (l_b^2-1, 1), \ldots\}_{k+1}.$$

New input enters the buffer:

a enters: $S_2^a = \{\ldots, (l_a^2+1, (i_a+1)\ mod\ m), (l_b^2-1, 1), \cdots\}_k$

b enters: $S_2^b = \{\ldots, (l_a^2, (i_a+1)\ mod\ m), (l_b^2, 1), \cdots\}_k$

j, j $\neq$ a,b enters: $S_2^j = \{\ldots, (l_a^2, (i_a+1)\ mod\ m), (l_b^2-1, 1), \cdots\}_k$

By the induction hypothesis:

$$EC_{\max}(S_1^a)_k < EC_{\max}(S_2^a)_k$$

$$EC_{\max}(S_1^b)_k < EC_{\max}(S_2^b)_k$$

$$EC_{\max}(S_1^j)_k < EC_{\max}(S_1^j)_k \qquad \forall j \neq a, b$$

Therefore:

$$EC_{\max}(S_1)_{k+1} < EC_{\max}(S_2)_{k+1}$$

(4) $l_a^2 > l_b^1,\ i_a = 0,\ 0 < i_b < m$

Due to the induction hypothesis, module a should be scheduled in S_1 and S_2.

For state S_1, schedule module a in this sub-cycle,

$$\{\ldots,\ (l_a^1,0),\ (l_b^1,i_b),\ \cdots\}_{k+1}$$

$$=>\ \{\ldots,\ (l_a^1-1,1),\ (l_b^1,(i_b+1)\ mod\ m),\ \cdots\}_{k+1}$$

New input enters the buffer:

a enters: $S_1^a = \{\ldots,\ (l_a^1,\ 1),\ (l_b^1,\ (i_b+1)\ mod\ m),\ \cdots\}_k$

b enters: $S_1^b = \{\ldots,\ (l_a^1-1,\ 1),\ (l_b^1+1,\ (i_b+1)\ mod\ m),\ \cdots\}_k$

j, j$\neq$a,b enters: $S_1^j = \{\ldots,\ (l_a^1-1,\ 1),\ (l_b^1,\ (i_b+1)\ mod\ m),\ \cdots\}_k$

For state S_2, schedule module a in this sub-cycle,

$$\{\ldots,\ (l_a^2,0),\ (l_b^2,i_b),\ldots\}_{k+1}$$

$$=>\ \{\ldots,\ (l_a^2-1,\ 1),\ (l_b^2,(i_b+1)\ mod\ m),\ldots\}_{k+1}.$$

New input enters the buffer:

a enters: $S_2^a = \{\ldots,\ (l_a^2,\ 1),\ (l_b^2,\ (i_b+1)\ mod\ m),\ \cdots\}_k$

b enters: $S_2^b = \{\ldots,\ (l_a^2-1,\ 1),\ (l_b^2+1,\ (i_b+1)\ mod\ m),\ \cdots\}_k$

j, j$\neq$a,b enters: $S_2^j = \{\ldots,\ (l_a^2-1,\ 1),\ (l_b^2,\ (i_b+1)\ mod\ m),\ \cdots\}_k$

By the induction hypothesis:

$$EC_{\max}(S_1^a) \leqq EC_{\max}(S_2^b);$$

$$EC_{\max}(S_1^b) \leqq EC_{\max}(S_2^a);$$

$$EC_{\max}(S_1^j) \leqq EC_{\max}(S_2^j) \qquad \forall j \neq a,b$$

Therefore:

$$EC_{\max}(S_1)_{k+1} \leqq EC_{\max}(S_2)_{k+1}.$$

(5) $l_a^2 = l_b^1$ Both modules are not scheduled in the current sub-cycle.

With the similar reasons as in III(b)(1), there exists a module z which is scheduled in the current sub-cycle. Because of the symmetry between the states of modules a and b, by the induction hypothesis II(b),

$$EC_{\max}(S_1^a)_k = EC_{\max}(S_2^b)_k$$

$$EC_{\max}(S_1^b)_k = EC_{\max}(S_2^a)_k \text{ and}$$

$$EC_{\max}(S_1^j)_k = EC_{\max}(S_2^j)_k \qquad j \neq a, b$$

Therefore:

$$EC_{\max}(S_1)_{k+1} = EC_{\max}(S_2)_{k+1}.$$

(6) $l_a^2 = l_b^1$ There exists a module x such that i_x=0 and

$$l_a^2 > l_x > l_a^1 \quad \text{if } i_a = 0 \text{ or}$$

$$l_b^1 > l_x > l_b^2 \quad \text{if } i_b = 0.$$

For the first case,

$$S_1 = \{\ldots, (l_a^1, 0), (l_b^1, i_b), \ldots, (l_x, 0), \cdots\}_{k+1};$$

$$S_2 = \{\ldots, (l_a^2, 0), (l_b^2, i_b), \ldots, (l_x, 0), \ldots\}_{k+1}.$$

With a similar argument as in III(b)(2), suppose module x is not scheduled in both states and module a is scheduled. Due to the symmetry between the states of module a and b, and by the induction hypothesis II(b),

$$EC_{\max}(S_1 |\ a\ scheduled)_k = EC_{\max}(S_2 |\ a\ scheduled)_k.$$

However, due to the induction hypothesis, II(a), scheduling x in state S_1 would be better than scheduling a because $l_x > l_a^1$.

$$EC_{\max}(S_1 |\ x\ scheduled)_k < EC_{\max}(S_1 |\ a\ scheduled).$$

$$< EC_{\max}(S_2 |\ a\ scheduled)_k$$

Therefore:

$$EC_{\max}(S_1)_{k+1} < EC_{\max}(S_2)_{k+1}.$$

The other case, i.e., $l_b^1 > l_x > l_b^2$ and i_b=0 can be similarly proved.

(7) $l_a^2 = l_b^1$, $\quad 0 = i_a < i_b < m \ or\ 0 = i_b < i_a < m$

The proof is very similar to III(b)(3) and III(b)(4), except in this case, $l_a^2 = l_b^1$ and $l_a^1 = l_b^2$. Therefore, the states S_1 and S_2 are symmetric in the states of the modules a and b. By the same argument as in the proof of

I(b)(3), the probability that $i_b=0$, $i_a<m$ is equals the probability that $i_a=0$, $i_b<m$. This implies:

$$EC_{\max}(S_1)_{k+1} = EC_{\max}(S_2)_{k+1}$$

From the above seven cases, it is seen that in all cases,

$$EC_{\max}(S_1)_{k+1} \leqq EC_{\max}(S_2)_{k+1}.$$

Therefore, by induction, part (b) of the theorem is proved. Because part (a) of the theorem utilizes the result of part (b) of the theorem, part (a) of the theorem is proved.

Q.E.D.

The above theorem has demonstrated that algorithm MWFMF is optimal in the sense that it minimizes the average completion time for a fixed set of random requests. Intuitively, algorithm MWFMF is better because it tries to keep all the modules as busy as possible. Suppose that some of the modules are requested more often than others. The requests to these more frequently requested modules become a bottleneck to the system whatever scheduling algorithms are used. However, a better scheduling algorithm should make use of the free cycles to schedule some requests for the less popular modules so that these requests would not accumulate after the processing of the more popular requests. This is the deficiency that occurs in other algorithms and is overcome by the MWFMF algorithm.

In addition to proving that the MWFMF algorithm has the best average case behavior, it may be necessary to show that the algorithm also possess the best best-case behavior and the best worst-case behavior. However, in this case, the best-case and the worst-case behavior are identical for all algorithms. The best-case behavior occurs when all the requests are made in a sequential order, that is, 0, 1, ..., m-1, 0, 1, ..., m-1, etc. No contention would occur and the throughput of the memory is maximized, that is, 1 request serviced every sub-cycle. On the other hand, the worst case behavior occurs when all the requests are directed to a single module. In this case, the bottleneck is at this module and the throughput of the memory is 1 request serviced every m sub-cycles. Algorithm MWFMF is better than other algorithms because it has a better average case behavior even though its best- and worst- case behavior are identical to the other algorithms.

Although the expected maximum completion time of the algorithm is minimized, it is not possible to make a similar conclusion as in Lemma 4.2 that the expected throughput of the memory is maximized because in this case, there is no relation between the expected maximum completion time and the expected throughput of the system. Furthermore, it is not useful to prove a similar theorem for the $\sum C_j$ case as in Lemma 4.3 because it is unclear that the objective of minimizing $\sum E(C_j)$ will be of any meaningful value.

Although Theorem 1 establishes the fact that the MWFMF algorithm is optimal, no throughput values are obtained analytically. In the next two sections, the throughput of the system is evaluated by using two techniques, embedded Markov Chains and simulations.

4.4.6 *Embedded Markov Chain Technique*

By assuming a saturated request rate, with inter-arrival time a constant multiple of the memory sub-cycle and a request queue with random requests, the two organizations can be analyzed by embedded Markov Chain technique [FEL50]. With a *RR scheduling algorithm*, a state of the system for Organization I is defined as $\{S_T, S_1, S_2, ..., S_b, S_m\}$, $1 \leq S_T, S_1, S_2, \ldots, S_b, S_m \leq m$ where $S_T, S_1, S_2, \ldots, S_b$ are the states of the b+1 buffers and S_m is the memory module that is being initiated in the current memory sub-cycle. The state of a buffer is the module number that the request in it wants to access. The number of states is therefore finite. A similar state can be defined for Organization II. It is obvious that the conditional probability of any future event, given the past event and the present state, is independent of the past event, that is, it satisfies the Markovian property.

$$P\{X_{n+1}=i_{n+1}|X_0=i_0, X_1=i_1, \dots, X_{n-1}=i_{n-1}, X_n=i_n\}$$
$$=P\{X_{n+1}=i_{n+1}|X_n=i_n\}$$
$$where\ n = 0, 1, 2, \dots$$

X_n = state of the system at the n-th transition

It is noticed that the Markovian property possessed by the two organizations is independent of n. Such a Markov chain is stationary. Let:

$$P_{i,j} = P\{X_{n+1}=i_{n+1} \mid X_n=i_n\}$$

Further, the time between successive transitions is constant and equals to the duration of the memory sub-cycle. This is called an embedded Markov Chain. The analysis of embedded Markov Chains is similar to that of Markov chains.

For an irreducible, ergodic Markov chain [ROS76], there exists a unique stationary probability distribution $\pi = \{\pi_j, j=1,2,\dots,n\}$ such that:

$$\pi_j = \sum_{i=1}^{n} \pi_i P_{i,j}$$

and

$$\sum_{i=1}^{n} \pi_i = 1$$

Using the matrix notation, it becomes

$$\pi = \pi P \qquad (4.1)$$

where:

$\pi = \{ \pi_1, \pi_2, \dots, \pi_n\}$

P = $\{P_{i,j}\}$, the transition matrix

n = the number of states in the system

The Markov Chain used to model the interleaved memory system is irreducible and positive recurrent because the chain is finite and all states communicate with each other. However, this chain is not ergodic because the period of the chain equals m. In this case, some of the conditions of the ergodic Markov Chains are weakened, but vector π still represents the unique fixed probability vector of P [KEM65]. Since the evaluation of the throughput only requires the use of the vector π, the technique for evaluating π in ergodic Markov Chains can still be applied here. This technique is illustrated in the following two examples.

Example 1

Consider Organization I with the following attributes:

m = 2

b = 1

scheduling algorithm - RR

access pattern - random

A state of the system is defined as $\{S_T, S_1, S_m\}$ where S_T is the state of B_T, S_1 is the state of B_1 and S_m is the current module that the system is initiating. The number of states can be reduced in half by considering only states in which $S_1 \leqq S_T$ and treating states in which $S_T < S_1$ the same as states in which $S_1 \leqq S_T$. The transition matrix is defined as

$$P = \begin{array}{cc} \begin{array}{cccccc} \{1,1,1\} & \{1,1,2\} & \{1,2,2\} & \{1,2,1\} & \{2,2,1\} & \{2,2,2\} \end{array} & \\ \left[\begin{array}{cccccc} 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0 \end{array}\right] & \begin{array}{c} \{1,1,1\} \\ \{1,1,2\} \\ \{1,2,2\} \\ \{1,2,1\} \\ \{2,2,1\} \\ \{2,2,2\} \\ \{2,2,2\} \end{array} \end{array}$$

On the first row, only the transitions from state $\{1,1,1\}$ to states $\{1,1,2\}$ and $\{1,2,2\}$ have non-zero probabilities. The state $\{1,1,1\}$ means that currently module 1 is initiated and the requests in both buffers B_T and B_1 are requesting module 1. Therefore, the request in B_1 can be satisfied. The content of B_T is moved into B_1, and a new address is accepted into the memory. Since the access pattern is random, this new request can be directed to either module 1 or 2 which results in states {1,1,2} or {1,2,2}. Note that S_m has changed from 1 to 2 because during the next memory sub-cycle, module 2 will be initiated. The other rows of the transition matrix can be interpreted similarly.

Solving the equation $\pi = \pi P$, we get:

$\pi = \{0.2, 0.1, 0.2, 0.2, 0.1, 0.2\}$

The utilization of the memory can be found by defining a new random variable

$$e_{\{S_T,S_1,S_m\}} = \begin{cases} 1 & S_T{=}S_m \textit{ or } S_1{=}S_m \textit{ or both} \\ 0 & \textit{otherwise} \end{cases}$$

$e_{\{S_T,S_1,S_m\}}$ equals 1 whenever during state $\{S_T,S_1,S_m\}$, one request is satisfied because there is a request in the buffers which requests a currently initiated module. For our example, the transpose of e is:

$e^T = \{1\ 0\ 1\ 1\ 0\ 1\}$

The utilization of the memory is π.e = 4/5 = 0.8. The bandwidth of the memory system is 0.8 * 2 = 1.6 words/memory cycle.

Example 2

Consider again Organization I with the following attributes:

m = 2

b = 1

Scheduling algorithm - MWFMF

Access pattern - random

The state space in this case is larger than the state space of the corresponding model with a RR scheduling algorithm because the next module to be initiated is determined dynamically and therefore the states of all modules must be known at all times. The objective of introducing a more complex algorithm like MWFMF is to initiate any free module with queued requests without constrains on the order of initiation. However, in this case, with m=2, improvement cannot be accomplished. Let us assume that module 1 is initiated during the current memory sub-cycle. In the next sub-cycle, module 1 cannot be initiated again because it has not finished its cycle. The only possibility is to initiate module 2. If all the requests in the buffers are requesting module 1, then no module is initiated in this sub-cycle and in the next sub-cycle, module 1 will be initiated again. The resulting sequence of initiation is the same as a model with an RR scheduling algorithm. Therefore the utilization of the model is the same for both algorithms when m=2. This happens because the maximum number of free modules is one in this special case. For $m \geqq 2$, the utilization for an MWFMF algorithm is higher because the maximum number of free modules is greater than one and the order of initiation is not necessary the same as the RR algorithm.

Let us complete this example by setting up the state space of the model. We must know at the beginning of each memory sub-cycle which modules are in service and what are the remaining service times that these modules need. We must also know the contents of the buffers. A state of the system is defined as $\{M_0, M_1, B_1, B_T\}$. M_0 is the module number of a module that is initiated 2 cycles ago and has finished its service at this time. M_1 is the module number of a module that is initiated 1 cycle ago and still needs 1 more cycle to finish its service. A value of 0 for M_0 or M_1 indicates that no module was initiated. B_1 and B_T are the states of the buffers and as in the last example, we consider only states with $B_1 \leqq B_T$. We have the following ranges of values, $0 \leqq M_0, M_1 \leqq 2$; $1 \leqq B_1, B_T \leqq 2$. The total number of states is 3*3*2*2 = 36. However, not all states are possible. For example, state {2,2,X,X} is not possible because it indicates that module 2 was initiated twice and simultaneously in the last two cycles. Another example of impossible state is {2,0,1,X}. This state indicates that no module was initiated in the last cycle (M_1=0), and the current contents of the buffers have a request for module 1. Since no new request was accepted in the last cycle, this request for module 1 must have existed in the previous cycle and therefore should have been initiated. By eliminating all these impossible states, we get a state space of 12 states: {0,1,1,1}, {0,1,1,2}, {1,0,1,1}, {1,2,1,1}, {1,2,1,2}, {1,2,2,2}, {0,2,2,2}, {0,2,1,2}, {2,0,2,2}, {2,1,2,2}, {2,1,1,2} and {2,1,1,1}. Solving the equation $\pi=\pi P$, we get

$$\pi=\{\frac{1}{20},\frac{1}{20},\frac{1}{10},\frac{1}{10},\frac{3}{20},\frac{1}{20},\frac{1}{20},\frac{1}{20},\frac{1}{10},\frac{1}{10},\frac{3}{20},\frac{1}{20}\}$$

By defining e^T as

$$e^T = \{0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1\}$$

The utilization of the modules is $\pi.e$ = 0.8

The transition matrices P used in Eq. 4.1 are large sparse matrices. The memory space required to store P is therefore substantially less. However, one big disadvantage about this approach is that the number of states is large. Although they can be reduced by eliminating duplicate or impossible states, the memory size and the computer time required for solution is still beyond the present computers' capability. For example, with 16 degrees of interleaving and b=2 (3 request buffers), the number of states for Organization I with an RR scheduling algorithm is 13056. This was calculated by treating permutations of the three buffers as equivalent states. With an RR scheduling algorithm, the module that the system currently initiates is sufficient to determine the next module to be initiated. With other scheduling algorithms, the number of states is more because the next module to be initiated is determined dynamically, and therefore the states of all the modules (whether they are busy or free) must be known at all times. However, regardless of the scheduling algorithm, the number of equivalent states can be reduced by a factor of m by noting that if a constant is added (modulo m) to each state variable, then the new state obtained must have the same stationary probability, and that the corresponding transition probability must also be the same. For the RR example given above, the 13056 states would be reduced to 816. Although the number of states is reduced, a solution using embedded Markov chains is still not practical. Since none of the states in the matrix are equivalent and therefore cannot be combined together, approximation techniques can be employed to reduce the number of states further. In [WAH76], an approximate embedded Markov chain solution for Organization I with RR scheduling algorithm is presented. The approximation is done by combining some states of the transition matrix into a single state when their transition probabilities into another state are "approximately" equal for all the states in the group. However, the difference between the approximate and the exact solutions are sometimes large. Moreover, the time it takes to generate the approximate matrix is still exponential because the transition probabilities of a state must be generated first before it can be determined whether the state can be combined with another state. The analytical solution using embedded Markov Chains is therefore not practical. In the next section, the solution using simulations is presented.

4.4.7 *Simulation Technique*

4.4.7.1 *Simulation Results*

Due to the difficulties mentioned in the last section, our evaluations are based on simulations. The simulations are run on a CDC 6400 computer. The simulation program was written in Fortran and the total time to generate all the results took over 12 hours on the CDC 6400.

Table 4.1 shows the results of simulation runs on Organization I for the memory utilization and the average waiting cycles where a *waiting cycle* is defined similar to Flores [FLO64] as the ratio of the waiting time and the memory cycle time. Two types of request sequences are considered, one in which the requests are generated randomly, and one in which the requests are derived directly from the execution trace of a program. The traces used have a size of 500,000 and were obtained by running a scientific Fortran program derived from BMD applications on a CDC 7600 and and they personify program characteristics of scientific applications. They have the following characteristics.

Table 4.1a Simulation Results for Organization I with RR Scheduling Algorithm (95% confidence interval shown assuming normal distribution)

		Random Request Model		Trace Driven Model	
m	b	E(Memory Utilization)	E(Waiting Cycles)	E(Memory Utilization)	E(Waiting Cycles)
2	0	0.668±0.0	1.75±0.0	0.727±0.003	1.69±0.0
	1	0.801±0.017	2.25±0.04	0.882±0.003	2.13±0.01
	2	0.858±0.001	2.75±0.0	0.928±0.003	2.62±0.33
	3	0.890±0.004	3.25±0.02	0.960±0.004	3.08±0.52
4	0	0.401±0.004	1.62±0.01	0.472±0.026	1.53±0.02
	1	0.565±0.015	1.89±0.02	0.636±0.043	1.79±0.07
	2	0.667±0.009	2.12±0.02	0.732±0.050	2.03±0.14
	3	0.726±0.007	2.38±0.02	0.825±0.059	2.21±0.23
8	0	0.222±0.002	1.56±0.0	0.276±0.026	1.45±0.06
	1	0.363±0.006	1.69±0.01	0.432±0.041	1.58±0.07
	2	0.461±0.005	1.81±0.01	0.525±0.049	1.72±0.10
	3	0.534±0.006	1.94±0.01	0.610±0.060	1.82±0.13
12	0	0.154±0.003	1.54±0.0	0.186±0.026	1.45±0.05
	1	0.266±0.005	1.63±0.01	0.306±0.042	1.55±0.10
	2	0.354±0.005	1.71±0.01	0.408±0.058	1.61±0.11
	3	0.423±0.008	1.79±0.01	0.484±0.070	1.69±0.10
16	0	0.117±0.002	1.53±0.01	0.157±0.015	1.40±0.05
	1	0.209±0.003	1.60±0.01	0.254±0.024	1.49±0.05
	2	0.285±0.003	1.66±0.01	0.345±0.033	1.54±0.06
	3	0.350±0.004	1.71±0.01	0.412±0.039	1.61±0.09

Table 4.1b Simulation Results for Organization I with FFF Scheduling Algorithm (95% confidence interval shown assuming normal distribution)

		Random Request Model		Trace Driven Model	
m	b	E(Memory Utilization)	E(Waiting Cycles)	E(Memory Utilization)	E(Waiting Cycles)
2	0	0.501±0.0	2.00±0.0	0.571±0.002	1.88±0.0
	1	0.668±0.014	2.50±0.05	0.789±0.003	2.27±0.11
	2	0.750±0.001	3.00±0.0	0.865±0.003	2.73±0.34
	3	0.802±0.003	3.50±0.02	0.924±0.003	3.17±0.53
4	0	0.289±0.003	1.86±0.01	0.316±0.018	1.79±0.04
	1	0.407±0.011	2.23±0.04	0.476±0.027	2.05±0.12
	2	0.489±0.007	2.53±0.04	0.600±0.041	2.25±0.25
	3	0.544±0.008	2.84±0.06	0.678±0.048	2.48±0.42
8	0	0.173±0.002	1.72±0.01	0.184±0.017	1.68±0.06
	1	0.264±0.004	1.95±0.02	0.304±0.029	1.82±0.12
	2	0.330±0.005	2.14±0.03	0.379±0.037	1.99±0.16
	3	0.378±0.003	2.32±0.03	0.441±0.043	2.13±0.20
12	0	0.126±0.002	1.66±0.01	0.147±0.023	1.57±0.06
	1	0.201±0.003	1.83±0.01	0.235±0.033	1.71±0.11
	2	0.258±0.002	1.97±0.02	0.305±0.042	1.82±0.13
	3	0.303±0.004	2.10±0.03	0.365±0.051	1.91±0.17
16	0	0.100±0.001	1.63±0.01	0.106±0.010	1.59±0.05
	1	0.163±0.002	1.77±0.01	0.187±0.017	1.67±0.08
	2	0.211±0.003	1.89±0.01	0.256±0.024	1.73±0.10
	3	0.252±0.003	1.99±0.02	0.314±0.030	1.80±0.13

Table 4.1c Simulation Results for Organization I with MWFMF Scheduling Algorithm (95% confidence interval shown assuming normal distribution)

		Random Request Model		Trace Driven Model	
m	b	E(Memory Utilization)	E(Waiting Cycles)	E(Memory Utilization)	E(Waiting Cycles)
2	0	0.667±0.008	1.75±0.01	0.727±0.003	1.69±0.0
	1	0.800±0.0	2.25±0.0	0.882±0.003	2.13±0.11
	2	0.859±0.003	2.75±0.03	0.928±0.003	2.62±0.33
	3	0.888±0.001	3.25±0.02	0.960±0.003	3.08±0.04
4	0	0.479±0.003	1.52±0.0	0.515±0.029	1.49±0.02
	1	0.612±0.003	1.82±0.01	0.673±0.043	1.74±0.08
	2	0.691±0.004	2.09±0.02	0.776±0.053	1.97±0.18
	3	0.740±0.004	2.35±0.04	0.831±0.059	2.20±0.28
8	0	0.355±0.002	1.35±0.01	0.385±0.038	1.33±0.06
	1	0.466±0.002	1.54±0.01	0.533±0.052	1.47±0.08
	2	0.544±0.004	1.69±0.01	0.612±0.058	1.61±0.11
	3	0.597±0.005	1.84±0.02	0.686±0.068	1.73±0.16
12	0	0.295±0.002	1.28±0.0	0.330±0.052	1.23±0.05
	1	0.399±0.003	1.42±0.01	0.472±0.066	1.35±0.08
	2	0.475±0.003	1.53±0.01	0.533±0.079	1.45±0.10
	3	0.524±0.002	1.64±0.01	0.614±0.088	1.54±0.12
16	0	0.259±0.001	1.24±0.0	0.300±0.028	1.21±0.05
	1	0.357±0.003	1.35±0.01	0.416±0.040	1.30±0.07
	2	0.424±0.002	1.44±0.01	0.511±0.049	1.37±0.08
	3	0.476±0.002	1.53±0.01	0.570±0.055	1.44±0.10

fraction of instruction word fetches	0.597
fraction of data word fetches	0.336
fraction of data word stores	0.067
average number of accesses per inst. executed	0.600
number of instructions per instruction word	2.787
fraction of instructions that need data	0.242
fraction of instructions that are	
unconditional jumps	0.044

successful conditional jumps		0.030
unsuccessful conditional jumps		0.015
number of instructions executed between		
conditional jumps	mean	22.3
	st'd dev.	10.3
unconditional jumps	mean	22.8
	st'd dev.	24.7
successful conditional jumps	mean	33.9
	st'd dev.	19.2
all dependent events	mean	11.4
(cond. + uncond. jumps)	st'd dev.	10.1

In Table 4.2, the simulation results for Organization II are shown. Since the existence of multiple sets of buffers allows a request at B_T to be blocked by a set of full buffers in a module while buffers of other modules may be empty, a column has been included in Table 4.2 to show the buffer utilization (this excludes the buffer B_T). The queue utilization results shown in Table 4.2 are normalized with respect to the buffer size b.

4.4.7.2 *Application of Multiple Linear Regression to Obtain a Closed Form Formula*

Using the results of the simulations and the assumption that the utilization is approximately 1 when b>>m (e.g. b=100, m=4), multiple linear regression is applied to fit a curve to the results [DRA66]. Based on the tail area of the partial F-value for testing the null hypothesis that a regression coefficient is zero, some of the terms in the polynomial have been eliminated. In Table 4.3, the coefficients for the regression analysis on the utilization and the waiting cycles of the two organizations under MWFMF scheduling algorithm are shown. The errors in the estimation can be shown to be less than 4% in most cases except

Table 4.2a Simulation Results for Organization II with RR scheduling Algorithm (95% confidence interval shown assuming normal distribution)

		Random Request Model			Trace Driven Model		
m	b	E(Memory Utilization)	E(Waiting Cycles)	E(Buffer Utilization)	E(Memory Utilization)	E(Waiting Cycles)	E(Buffer Utilization)
2	0	0.667±0.0	1.75±0.0	-	0.727±0.003	1.69±0.0	-
	1	0.801±0.004	2.50±0.02	0.700±0.006	0.882±0.003	2.43±0.16	0.760±0.049
	2	0.857±0.002	3.25±0.02	0.715±0.003	0.928±0.003	3.18±0.44	0.761±0.102
	3	0.890±0.003	4.00±0.01	0.732±0.004	0.960±0.003	3.92±0.61	0.768±0.142
4	0	0.401±0.004	1.62±0.01	-	0.472±0.026	1.53±0.02	-
	1	0.629±0.004	2.18±0.01	0.492±0.005	0.737±0.052	2.09±0.17	0.554±0.085
	2	0.731±0.005	2.75±0.03	0.515±0.009	0.847±0.058	2.70±0.36	0.597±0.136
	3	0.792±0.004	3.33±0.02	0.531±0.006	0.903±0.064	3.34±0.50	0.621±0.158
8	0	0.222±0.002	1.56±0.0	-	0.276±0.026	1.45±0.06	-
	1	0.487±0.006	1.97±0.01	0.347±0.006	0.586±0.055	1.87±0.11	0.384±0.063
	2	0.626±0.006	2.41±0.02	0.379±0.008	0.793±0.078	2.40±0.25	0.494±0.113
	3	0.705±0.004	2.86±0.04	0.397±0.006	0.862±0.085	2.95±0.45	0.518±0.148
12	0	0.154±0.003	1.54±0.0	-	0.186±0.026	1.45±0.05	-
	1	0.417±0.005	1.88±0.01	0.283±0.005	0.599±0.083	1.73±0.10	0.354±0.070
	2	0.569±0.007	2.26±0.02	0.318±0.007	0.733±0.105	2.22±0.26	0.404±0.125
	3	0.661±0.005	2.66±0.02	0.338±0.006	0.753±0.109	2.63±0.47	0.381±0.151
16	0	0.117±0.002	1.53±0.01	-	0.157±0.015	1.40±0.05	-
	1	0.379±0.005	1.82±0.01	0.249±0.004	0.502±0.048	1.73±0.09	0.303±0.049
	2	0.534±0.008	2.18±0.02	0.283±0.008	0.692±0.066	2.05±0.17	0.333±0.075
	3	0.626±0.004	2.54±0.08	0.300±0.005	0.745±0.072	2.23±0.32	0.284±0.094

Table 4.2b Simulation Results for Organization II with FFF Scheduling Algorithm (95% confidence interval shown assuming normal distribution)

		Random Request Model			Trace Driven Model		
m	b	E(Memory Utilization)	E(Waiting Cycles)	E(Buffer Utilization)	E(Memory Utilization)	E(Waiting Cycles)	E(Buffer Utilization)
2	0	0.501±0.0	2.00±0.0	-	0.571±0.002	1.88±0.0	-
	1	0.670±0.004	2.74±0.0	0.669±0.004	0.789±0.003	2.56±0.16	0.732±0.045
	2	0.751±0.003	3.50±0.02	0.688±0.010	0.865±0.003	3.30±0.48	0.743±0.097
	3	0.789±0.002	4.27±0.01	0.699±0.004	0.924±0.003	4.00±0.62	0.758±0.138
4	0	0.289±0.003	1.86±0.01	-	0.316±0.018	1.79±0.04	-
	1	0.458±0.005	2.54±0.01	0.454±0.007	0.557±0.042	2.38±0.22	0.519±0.095
	2	0.556±0.004	3.19±0.03	0.483±0.009	0.702±0.048	2.95±0.48	0.558±0.149
	3	0.628±0.003	3.80±0.04	0.503±0.007	0.801±0.057	3.54±0.72	0.596±0.184
8	0	0.173±0.002	1.72±0.01	-	0.184±0.017	1.68±0.06	-
	1	0.329±0.005	2.32±0.03	0.311±0.010	0.406±0.040	2.20±0.19	0.360±0.069
	2	0.436±0.002	2.88±0.02	0.348±0.004	0.604±0.060	2.73±0.36	0.461±0.121
	3	0.498±0.004	3.44±0.07	0.363±0.011	0.671±0.067	3.36±0.65	0.492±0.162
12	0	0.126±0.002	1.66±0.01	-	0.147±0.023	1.57±0.06	-
	1	0.278±0.004	2.20±0.03	0.250±0.007	0.396±0.055	2.03±0.18	0.325±0.080
	2	0.383±0.005	2.71±0.05	0.285±0.010	0.510±0.073	2.64±0.43	0.376±0.137
	3	0.457±0.004	3.22±0.05	0.310±0.009	0.529±0.085	3.19±0.55	0.358±0.752
16	0	0.100±0.001	1.63±0.01	-	0.106±0.010	1.59±0.05	-
	1	0.252±0.003	2.11±0.02	0.217±0.005	0.345±0.032	1.96±0.14	0.267±0.050
	2	0.349±0.003	2.61±0.03	0.250±0.005	0.478±0.046	2.33±0.30	0.286±0.082
	3	0.424±0.004	3.09±0.05	0.141±0.008	0.527±0.050	2.62±0.46	0.262±0.092

Table 4.2c Simulation Results for Organization II with MWFMF Scheduling Algorithm (95% confidence interval shown assuming normal distribution)

		Random Request Model			Trace Driven Model		
m	b	E(Memory Utilization)	E(Waiting Cycles)	E(Buffer Utilization)	E(Memory Utilization)	E(Waiting Cycles)	E(Buffer Utilization)
2	0	0.667±0.008	1.75±0.01	-	0.727±0.003	1.69±0.0	-
	1	0.799±0.003	2.50±0.01	0.700±0.005	0.882±0.003	2.43±0.16	0.760±0.049
	2	0.856±0.005	3.25±0.01	0.714±0.007	0.928±0.003	3.18±0.44	0.761±0.102
	3	0.890±0.020	4.00±0.02	0.724±0.006	0.960±0.003	3.92±0.61	0.768±0.142
4	0	0.419±0.003	1.52±0.0	-	0.515±0.029	1.49±0.02	-
	1	0.648±0.001	2.13±0.01	0.482±0.002	0.738±0.052	2.07±0.18	0.539±0.090
	2	0.743±0.003	2.72±0.01	0.515±0.005	0.838±0.059	2.70±0.36	0.588±0.135
	3	0.795±0.002	3.31±0.02	0.528±0.003	0.902±0.032	3.35±0.53	0.625±0.157
8	0	0.355±0.002	1.35±0.01	-	0.385±0.038	1.33±0.06	-
	1	0.534±0.005	1.85±0.01	0.328±0.007	0.624±0.062	1.84±0.13	0.398±0.077
	2	0.651±0.003	2.33±0.01	0.371±0.003	0.799±0.079	2.40±0.26	0.498±0.118
	3	0.717±0.003	2.81±0.02	0.391±0.004	0.849±0.083	2.95±0.44	0.510±0.144
12	0	0.295±0.002	1.28±0.0	-	0.365±0.052	1.23±0.05	-
	1	0.472±0.005	1.72±0.01	0.256±0.005	0.624±0.089	1.68±0.14	0.343±0.095
	2	0.602±0.004	2.16±0.01	0.308±0.004	0.735±0.106	2.19±0.29	0.395±0.135
	3	0.683±0.006	2.58±0.03	0.332±0.009	0.756±0.109	2.61±0.49	0.377±0.154
16	0	0.259±0.001	1.24±0.0	-	0.300±0.028	1.21±0.05	-
	1	0.439±0.006	1.64±0.01	0.217±0.007	0.565±0.054	1.62±0.10	0.289±0.061
	2	0.564±0.007	2.05±0.02	0.264±0.007	0.692±0.066	1.97±0.21	0.306±0.083
	3	0.647±0.003	2.44±0.02	0.290±0.004	0.745±0.072	2.17±0.34	0.271±0.096

Table 4.3 - Coefficients of 3rd Order Polynomial Regression of Organization I and II under MWFMF Scheduling Algorithm
(RRM - Random Request Model; TDM - Trace Driven Model)
Note: All other coefficients are set to zero.

Utilization

Model	m^2	m	1/m	$b^{1/3}$	$b^{1/2}$	$b^{1/4}$	$\frac{b^{1/3}}{m}$	$\frac{b^{1/2}}{m}$	const.
RRM-I	0.00050	-0.02011	0.58124	1.80176	-0.32495	-1.37185	0.27655	-0.21970	0.41273
TDM-I	0.00065	-0.02312	0.62605	2.29106	-0.45177	-1.69628	0.18115	-0.18268	0.45447
RRM-II	-0.00009	-0.00283	0.79465	3.04862	-0.64641	-2.17849	-0.22013	0.00866	0.26880
TDM-II	-0.00012	-0.00301	0.80863	2.72327	-0.61966	-1.80155	-0.44904	0.11118	0.33023

Waiting Cycles

Model	m^3	m^2	m^2b	m	b	mb	const.
RRM-I	-0.00109	0.03312	0.00314	-0.31779	0.61021	-0.08138	2.30046
TDM-I	-0.00100	0.03038	0.00308	-0.29282	0.56690	-0.07881	2.19725
RRM-II	-0.00082	0.02570	0.00219	-0.26252	0.84230	-0.06200	2.19883
TDM-II	-0.00075	0.02432	0.00016	-0.25363	0.77708	0.03024	2.12700

for a few cases with b=0, where the error gets to around 10%. From the polynomial equation we have obtained, we can extrapolate our results beyond b=3. The errors in extrapolating the values of utilization is small because the asymptotic value of utilization when b is large is known. However, the errors may be large when extrapolating the values of waiting cycle because its asymptotic values are not known. With Organization I, $e_B = 1$, and therefore the values of waiting cycles can be derived from the values of utilization by applying Little's Formula. With Organization II, $e_B < 1$, and the values of $u_{m,b}$ and $w_{m,b}$ must be known in order to estimate e_B. Since asymptotic values of $w_{m,b}$ and e_B do not exist, the errors may be large in this case.

In Figures 4.13 to 4.18, the performance of Organizations I and II are shown. The actual simulation results are used for $b \leq 3$ while extrapolations are made for b>3. In Fig. 4.13, a plot of the improvement in memory utilization with buffer size for Organization I with m = 8 is shown. It is seen that the improvement in memory utilization approaches a constant rate as the buffer size is increased. Further, the MWFMF algorithm gives the best performance. In Fig. 4.14, a plot of the expected waiting cycles for different buffer sizes of Organization I is shown for m = 8. It is seen that the increase of waiting cycles is much slower than the increase of buffer sizes and the increase is almost linear. The trace driven simulation results show a higher improvements in memory utilization and a smaller number of waiting cycles than the random request model. This is because there is a higher correlation between consecutive requests and the requests are likely to be made in a consecutive order. As a result, there is less contention in the system. The curves showing the estimated results due to dependencies are discussed in the following sections. The above observations are also true for other values of m. Further, the MWFMF algorithm has the minimum amount of waiting time among the three algorithms studied. In Figures 4.15 and 4.16, the decrease in memory utilization and waiting cycles for increasing degrees of interleaving of Organization I with a MWFMF algorithm are plotted. The rate of decrease in memory utilization is more pronounced and the utilization is higher when the degree of interleaving is small. Also, the effects on waiting cycles due to buffer size is very small when the ratio of buffer size to degrees of interleaving is small. Other scheduling algorithms also possess the same properties. The effects on the memory utilization and the waiting cycles for various buffer sizes of Mode II are similar to those of Organization I. In Fig. 4.17, the effects on buffer utilization are shown for various buffer sizes of Organization II. It is seen that the buffers are less utilized as the size is increased. This also accounts for the diminishing increase in memory utilization as the buffer size is increased. The difference in buffer utilization among the three scheduling algorithms is very small. However, extrapolations for values of b beyond 3 are not accurate

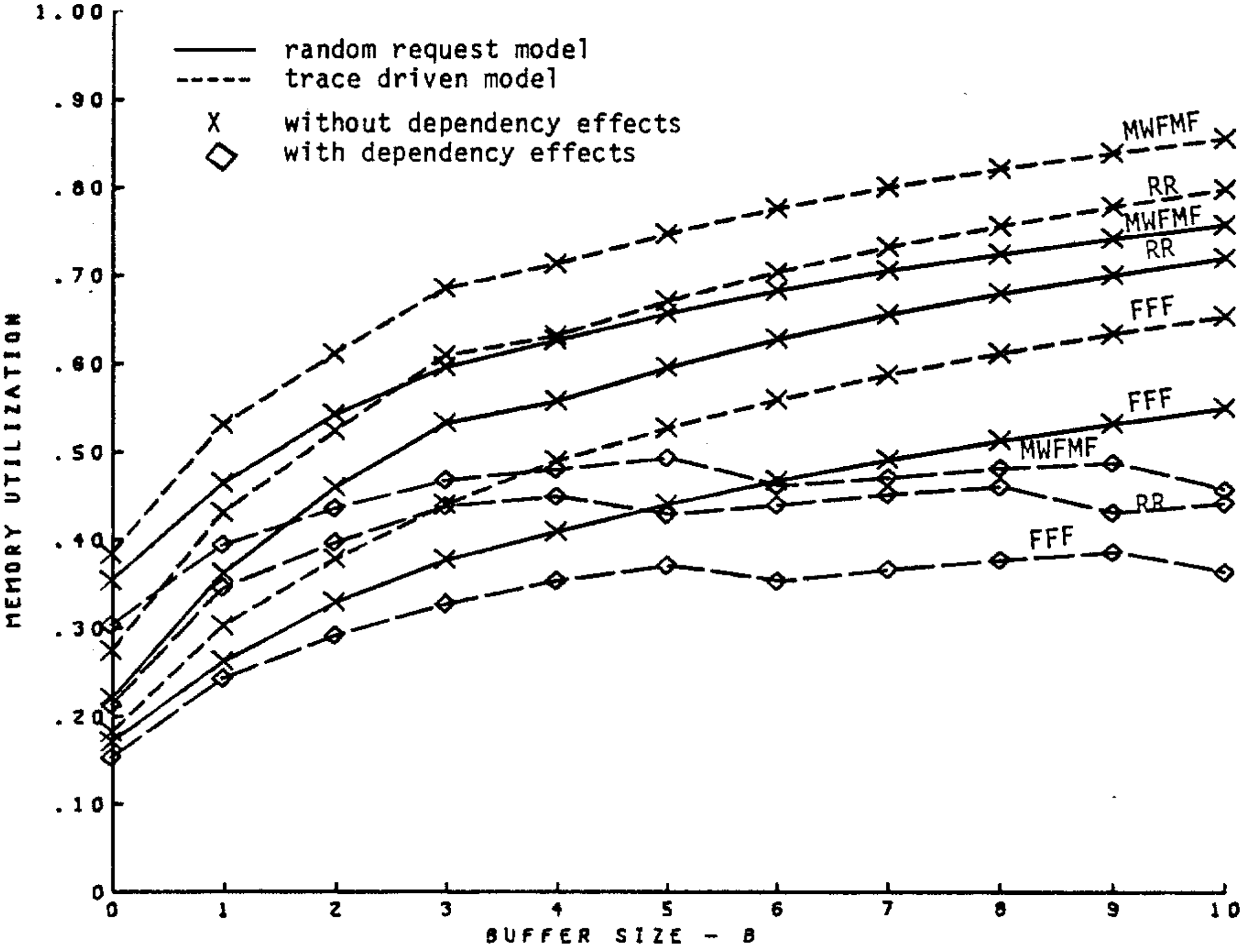


Figure 4.13 The Improvement of Average Memory Utilization with Buffer Size for Org. I (Degrees of Interleaving = 8)

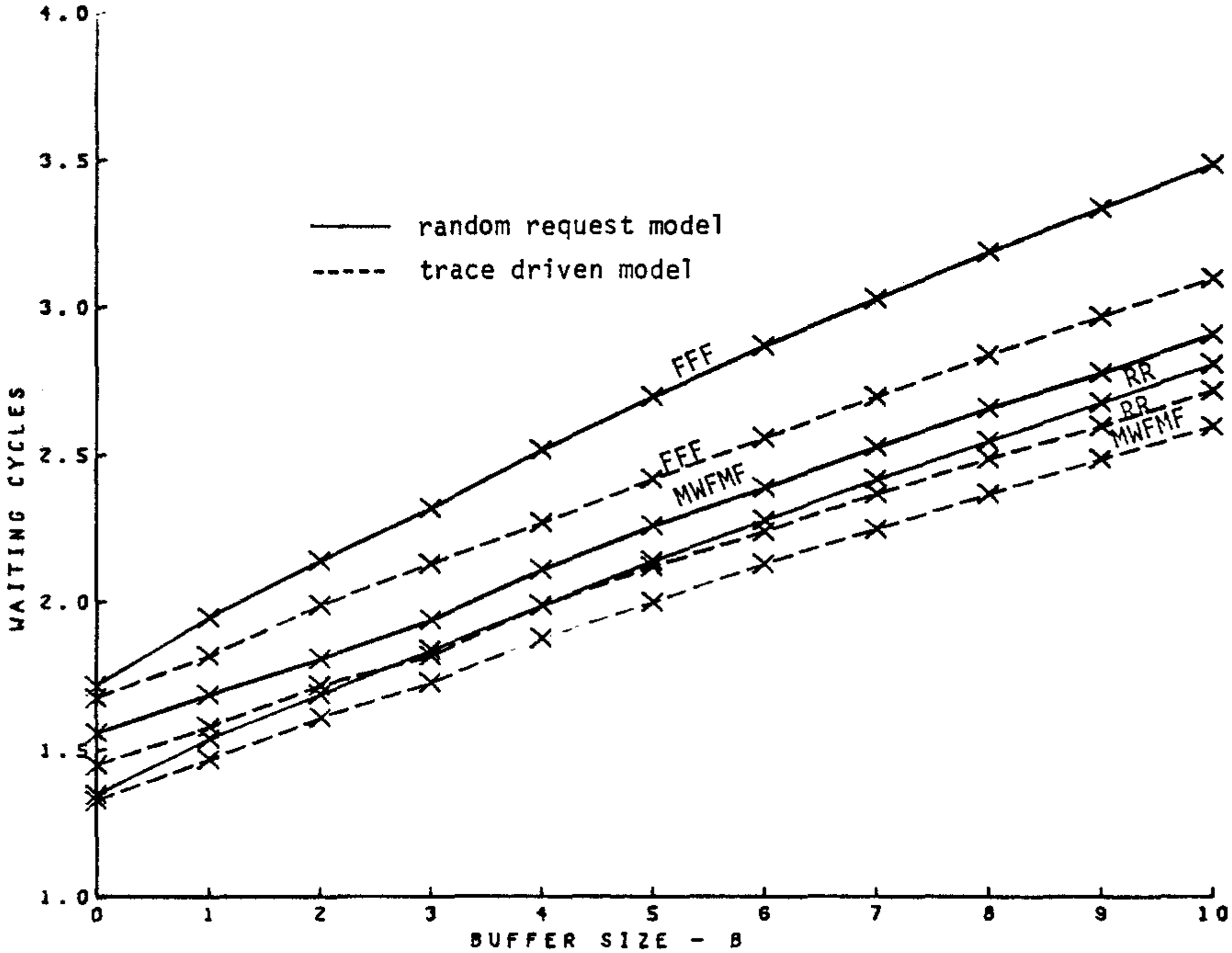


Figure 4.14 The Increase of Average Waiting Cycles with Buffer Size for Org. I (Degrees of Interleaving = 8)

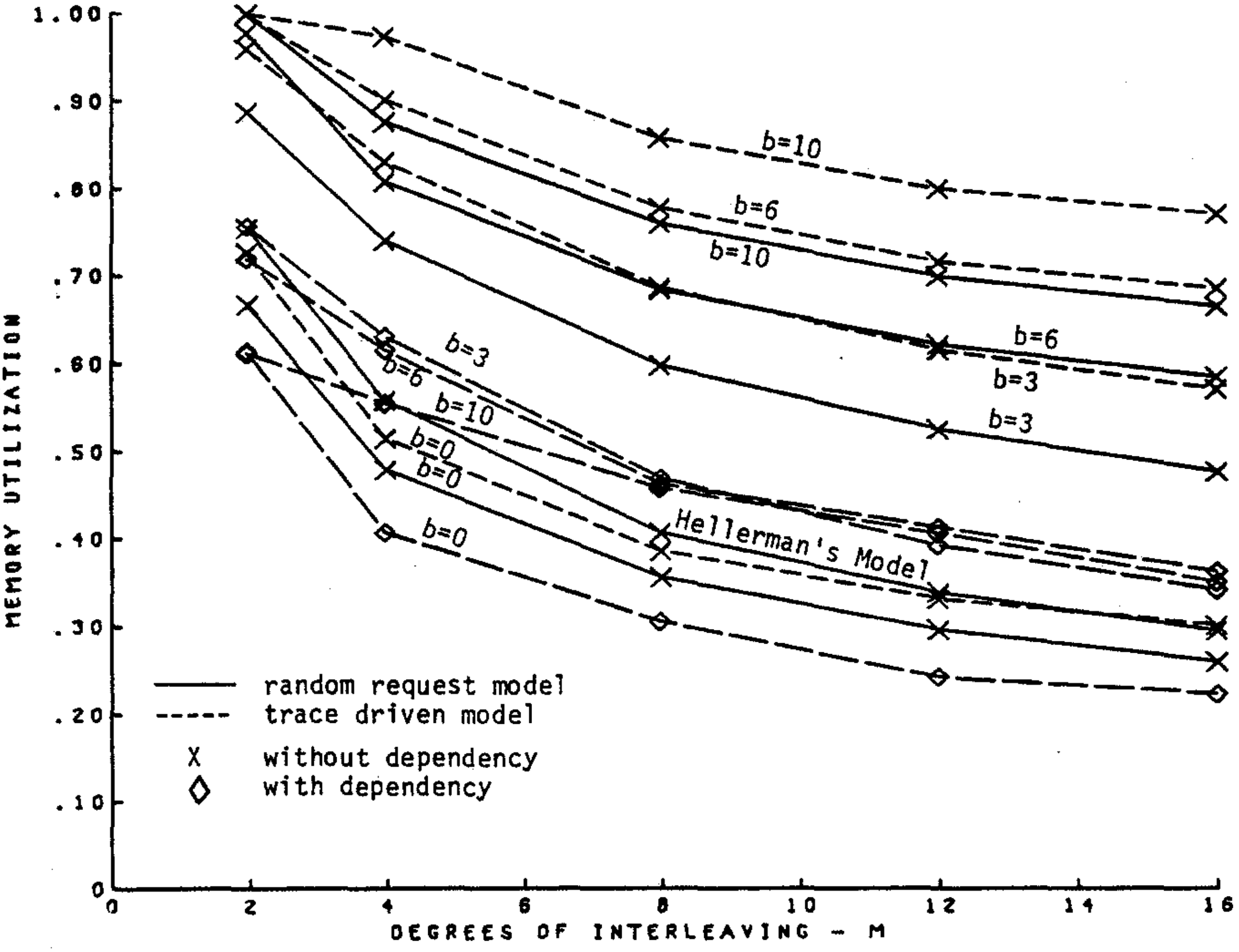


Figure 4.15 The Decrease of Average Memory Utilization with respect to Degrees of Interleaving for Org. I with MWFMF Scheduling Algorithm

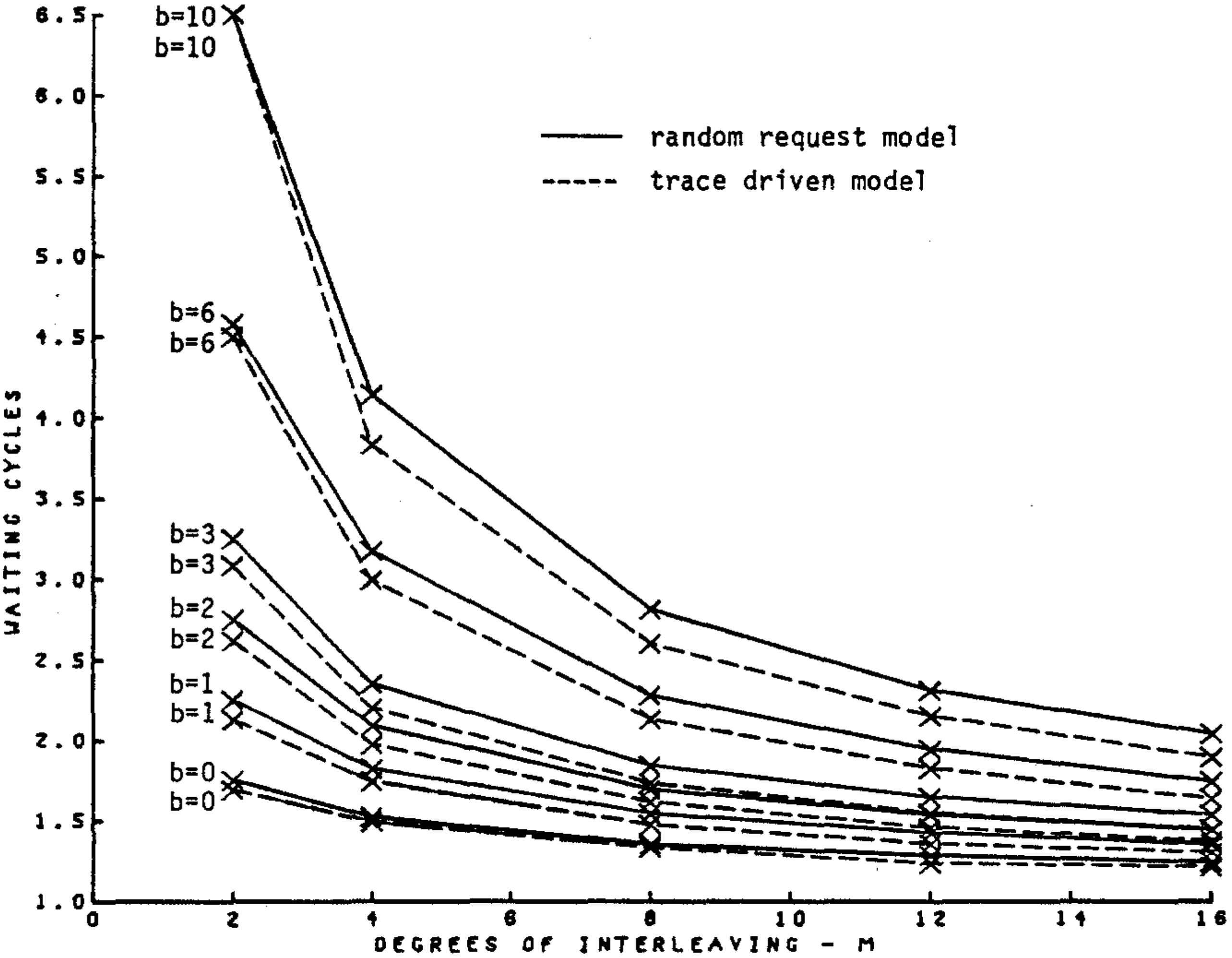


Figure 4.16 The Decrease of Waiting Cycles with respect to Degrees of Interleaving for Org. I with MWFMF Scheduling Algorithm

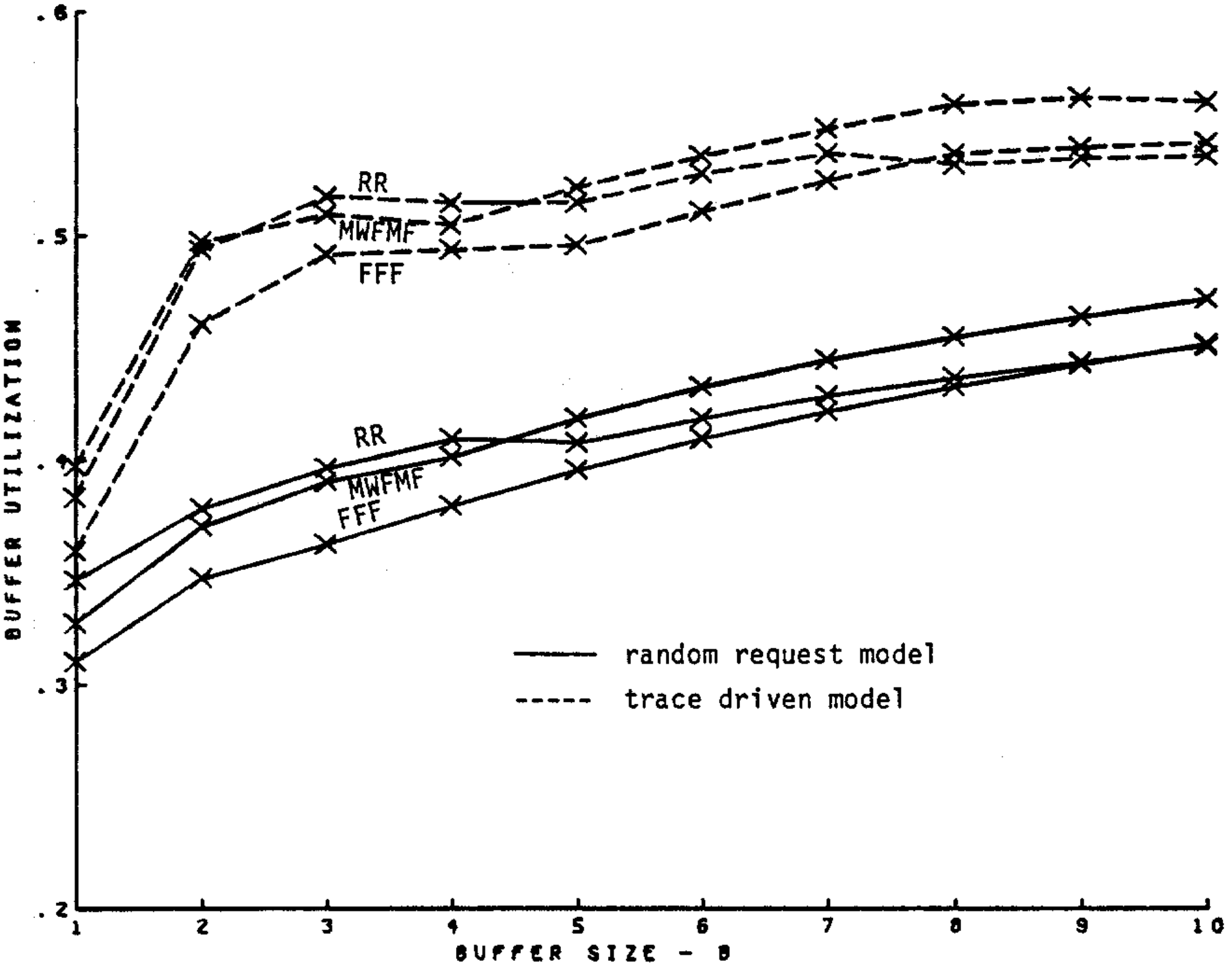


Figure 4.17 The Average Buffer Utilization for Org. II
(Degrees of Interleaving = 8)

for Organization II for reasons noted before. In Fig 4.18, a plot of buffer utilization versus different degrees of interleaving is shown. The buffer utilization drops as the number of modules is increased. However, it is seen in both Figures 4.17 and 4.18 that the buffer utilization is not sensitive to buffer size changes. The decrease in buffer utilization is due to the fact that there is a higher probability that B_T is blocked when the number of modules is increased.

4.4.8 *Effects of separating the instruction and the data area*

The previous results have been obtained from simulations using a merged instruction and data area. Since an instruction access results in some data accesses, it is desirable to place the data accessed in modules not conflicting with the next instruction accessed. This motivates us to investigate the separation of instruction and data area into different modules in the main memory. Sastry et. al. [SAS75] and Nutt [NUT77] have made some pioneering studies on the separation of instruction and data areas, but they have assumed a non-pipelined multi-processor system. We study the effects with respect to a pipelined processor here. In this section, an organization with separate instruction and data modules is compared against an organization with merged instruction and data modules using the traces available. Consecutive instruction words are put in consecutive instruction memories and consecutive data locations are put in consecutive data memories.

The characteristics of the traces reveal that 60% of the accesses are instructions and the rest are data accesses, therefore the modules should be divided according to this ratio approximately. Since it is desirable to have the number of instruction modules and the number of data modules an integral power of 2 for ease of address decoding, the modules are divided into a 4-2 partition so that four of the modules are instruction modules and the two are data modules. It is not possible to designate exactly 60% of the modules as

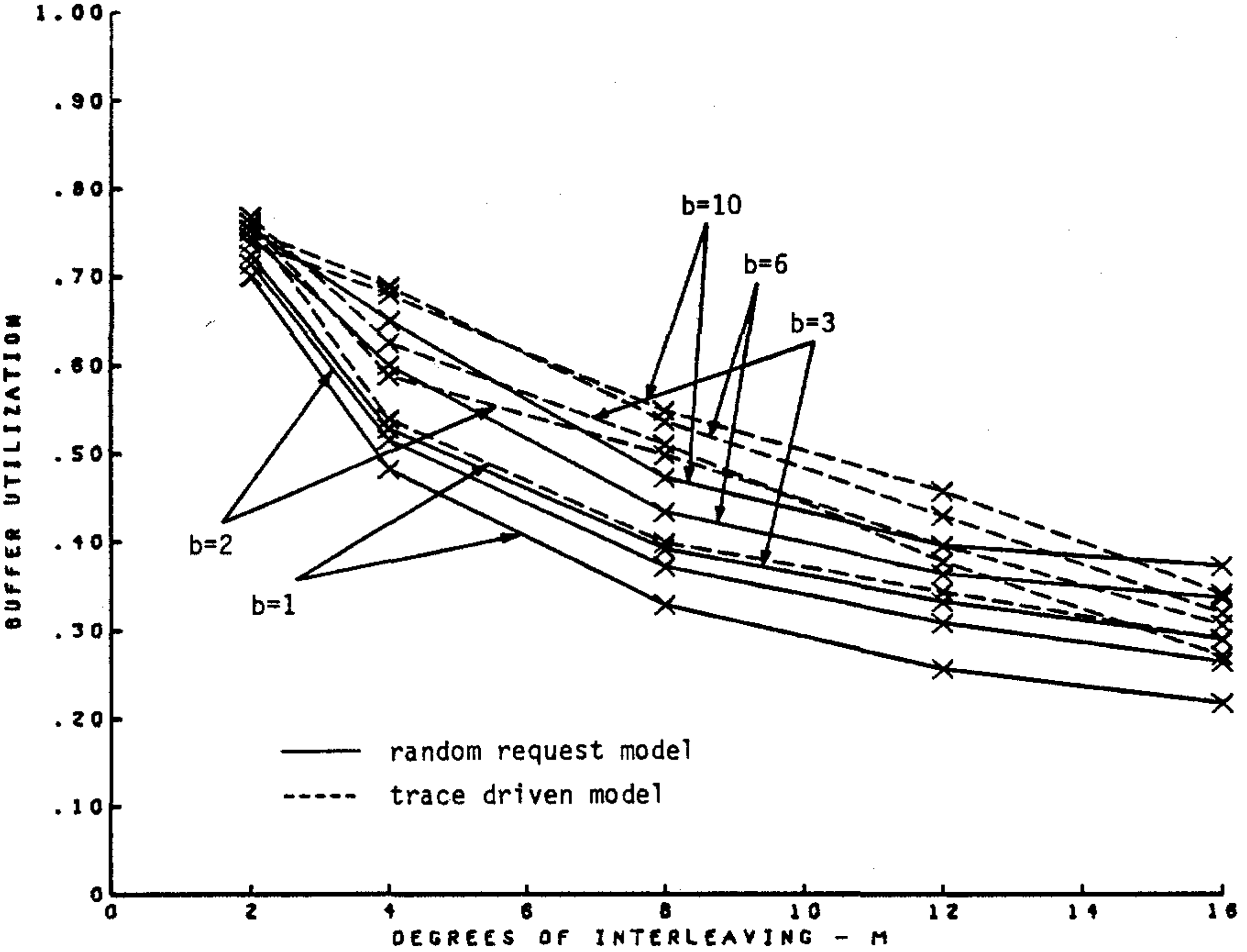


Figure 4.18 The Average Buffer Utilization versus the Degrees of Interleaving for Org. II with MWFMF Scheduling Algorithm (m=8)

instruction modules and to satisfy the requirement that the number be an integral power of 2. Since there are 6 modules in the 4-2 partition, it is necessary to compare the performance of the 4-2 partition against a hypothetical 6 way interleaved system with merged instruction and data modules. The results are shown in Tables 4.4 and 4.5. It is seen that the differences between the two alternatives are minimal. In fact in some cases, the merged model seems to

Table 4.4 Comparison between Merged and Separated Instruction- Data Areas for Organization I - Trace Driven Simulation.

			RR		FFF		MWFMF	
	m	b	Memory util.	Waiting cycles	Memory util.	Waiting cycles	Memory util.	Waiting cycles
Merged	6	0	0.338	1.49	0.243	1.69	0.459	1.36
Inst.-Data		1	0.501	1.65	0.403	1.83	0.624	1.53
Areas		2	0.657	1.76	0.479	2.04	0.695	1.72
(m=6)		3	0.726	1.92	0.543	2.23	0.752	1.89
Separate	6	0	0.336	1.50	0.270	1.62	0.486	1.34
Inst.-Data		1	0.517	1.64	0.394	1.85	0.619	1.54
Areas		2	0.618	1.81	0.484	2.03	0.692	1.72
(4-2 ways)		3	0.696	1.96	0.540	2.24	0.730	1.91

Table4.5 Comparison between Merged and Separated Instruction- Data Areas for Organization II - Trace Driven Simulations

			RR			FFF			MWFMF		
	m	b	Mem. Util.	Wait. Cycle	Buf. Util.	Mem. Util.	Wait. Cycle	Buf. Util.	Mem. Util.	Wait. Cycle	Buf. Util.
Merged	6	0	0.34	1.49	-	0.24	1.69	-	0.46	1.36	-
Inst.-Data		1	0.69	1.95	0.49	0.50	2.23	0.44	0.69	1.94	0.48
Areas		2	0.80	2.46	0.50	0.61	2.83	0.47	0.79	2.49	0.50
(m=6)		3	0.81	2.88	0.45	0.63	3.27	0.42	0.81	2.87	0.45
Sep.	6	0	0.34	1.50	-	0.27	1.62	-	0.49	1.34	-
Inst.-Data		1	0.65	1.98	0.47	0.49	2.18	0.40	0.67	1.90	0.43
Areas		2	0.75	2.42	0.45	0.57	2.74	0.41	0.76	2.41	0.45
(4-2 Ways)		3	0.77	2.96	0.45	0.59	3.29	0.40	0.77	2.95	0.45

perform a little better. This is due to the unequal utilization and waiting cycles of the modules in the separated case. From the simulation results on the utilization of the individual modules (not shown), the instruction modules are found to be under utilized while the data modules are found to be over utilized. One way to improve the performance of the system is to design the system with a good instruction-data access ratio so that the utilization of the instruction and the data modules are approximately equal and the number of instruction and data modules are integral powers of 2. However, this ratio is highly program dependent and is impossible to fix at the design stage. We conclude that the improvement due to separation is minimal for this architecture, the CDC 7600, and the specific class of programs.

4.4.9 *Degradation in Performance Due to Dependencies*

In the previous sections, we have simulated the organizations under the assumption that there is a high request rate from the pipe so that any empty buffers can be replenished until they are full or a blockage occurs. However, this assumption is not totally valid in a pipelined uni-processor. As mentioned earlier, there are three sources of interferences which result in emptying the pipe and reloading a new instruction stream. In the process of emptying the pipe, new memory requests are not generated and the memory becomes idle after all the pending requests are serviced. The utilization of the memory is therefore lower than our simulated results. One solution is to simulate a pipe together with the memory. However, different computers handle dependencies differently, and the simulation of a particular machine is too limited in scope. We therefore choose to estimate the resulting utilization with a general model.

4.4.9.1 *The Model Used to Estimate the Performance Due to Dependencies*

Without loss of generality, all dependencies can be represented as a successful (the jump is taken) or an unsuccessful conditional jump. In a conditional jump instruction, the condition code is set earlier by an instruction which may still be in the pipe. Until that instruction finishes and sets the condition code, the jump instruction cannot proceed. It is assumed that the pipe prefetches but does not decode the target instruction. If it is an unsuccessful jump, the pipe can proceed after the condition code has been set. If it is a successful jump, the pipe has to wait until both the condition code is set and the target instruction is fetched from the memory. An unconditional jump can be modelled as a successful conditional jump in which the condition code is available immediately. A register interlock is the same as an unsuccessful conditional jump instruction and an interrupt is the same as a successful conditional jump in which the entire pipe has to be emptied.

The model used in the estimation is shown in Fig. 4.19. A linear pipe is considered. The instruction prefetch unit has to fetch instructions ahead of the instruction decode unit so that the decode unit never has to wait for instruction fetches. Let

L = number of stages of the pipe;

T = time needed to pass through one a stage of the pipe;

f = the number of instruction words prefetched.

The memory is assumed to be a single server with a constant service time of rate $u_{m,b}$, and a finite buffer space of length M-$u_{m,b}$*m (Eq. 4.2, 4.3). The service discipline in the buffers is FIFO and the waiting time for a request is W (Eq. 4.4). Since we are interested in getting an expected value of the performance, the model is a sufficient approximation of the actual model. It is also assumed that the occurences of successive dependent requests are separated far enough and have no effect on each other. By "far enough", it is meant that after a dependency is resolved, sufficient time elapses so that all the buffers are filled up

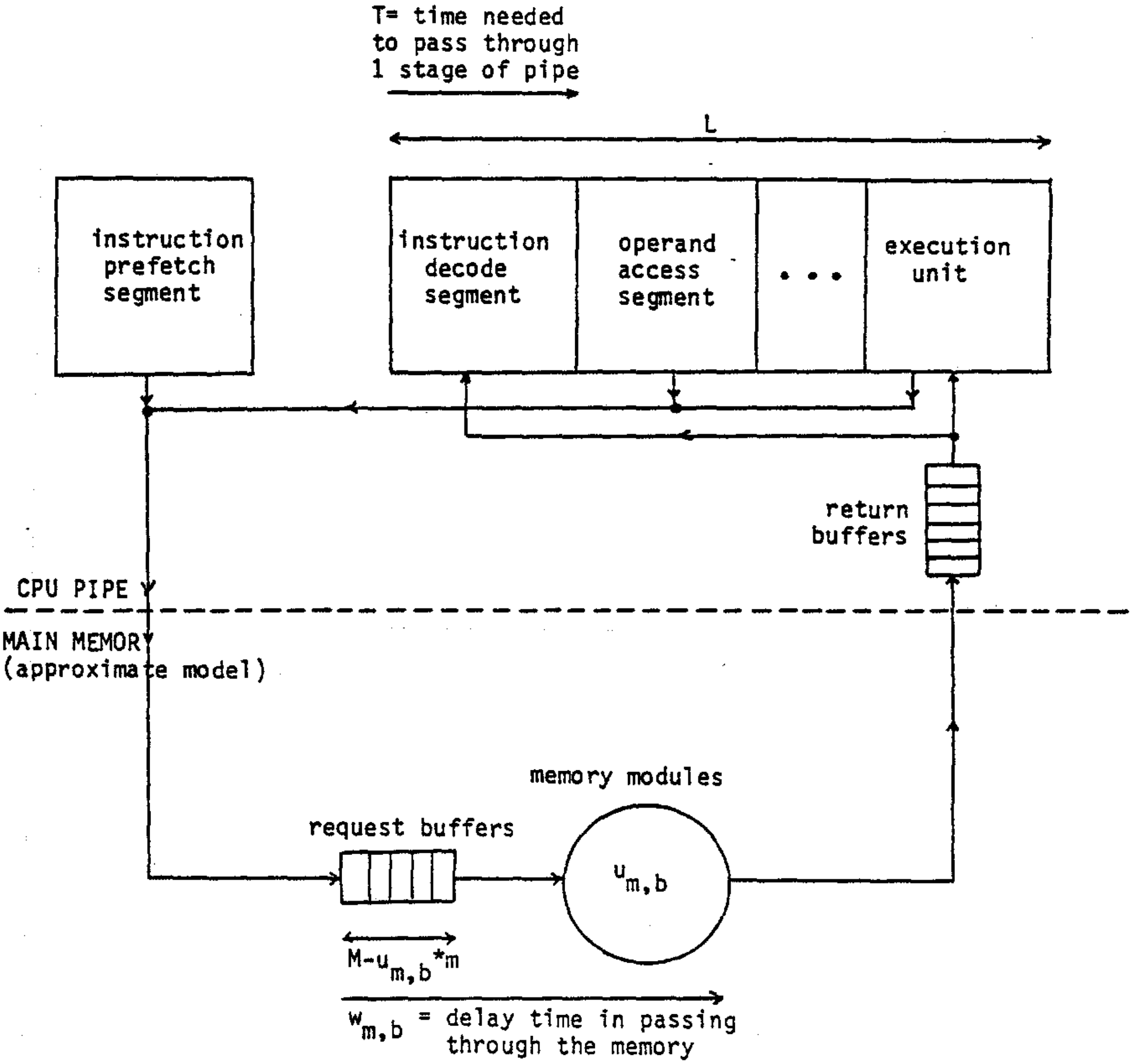

T= time needed to pass through 1 stage of pipe
L
instruction prefetch segment
instruction decode segment
operand access segment
• • •
execution unit
return buffers
CPU PIPE
MAIN MEMORY
(approximate model)
memory modules
request buffers
$u_{m,b}$
$M-u_{m,b}*m$
$w_{m,b}$ = delay time in passing through the memory

Figure 4.19 Model of CPU-Memory used for Estimation of the Effects of Dependency

before the occurence of another dependent request. The maximum time needed is $u_{m,b}*M$, (Fig. 4.22). This assumption is necessary because the effect due to each dependent request can be found separately and the overall effect due to all the dependent requests is the sum of the individual effects. From the statistics of the traces which are shown in Figures 4.20 and Fig. 4.21, it is found that successive dependent requests are separated by an average of 12 instructions. Successive dependent requests may therefore have effects on each other and our analysis slightly under-estimates the actual performance.

4.4.9.2 *Computation of Degradation in Performance*

The effect of dependencies is measured in terms of an idle period. An *idle period* of the memory is defined to be a time interval during which requests to the memory stop. The idle period is measured in terms of the number of memory sub-cycles. At the beginning of an idle period, the number of requests drops gradually to zero (Fig. 4.22). The resulting utilization of the module is lower as is evident from a similar model with a smaller buffer size. When the pipe starts requesting again, the number of requests in the buffers gradually builds up to the maximum amount. The idle period is defined this way because it represents an average length of the time during which the buffers are not fully utilized. Let

d = distance in terms of the number of pipe segments between the instruction setting the condition code and the conditional jump instruction at the decode segment;

r = average number of requests generated per instruction executed;

i = number of instructions per instruction word;

x^{s}_{CJ} = fraction of instructions executed that are successful conditional jumps;

x^{f}_{CJ} = fraction of instructions executed that are unsuccessful conditional

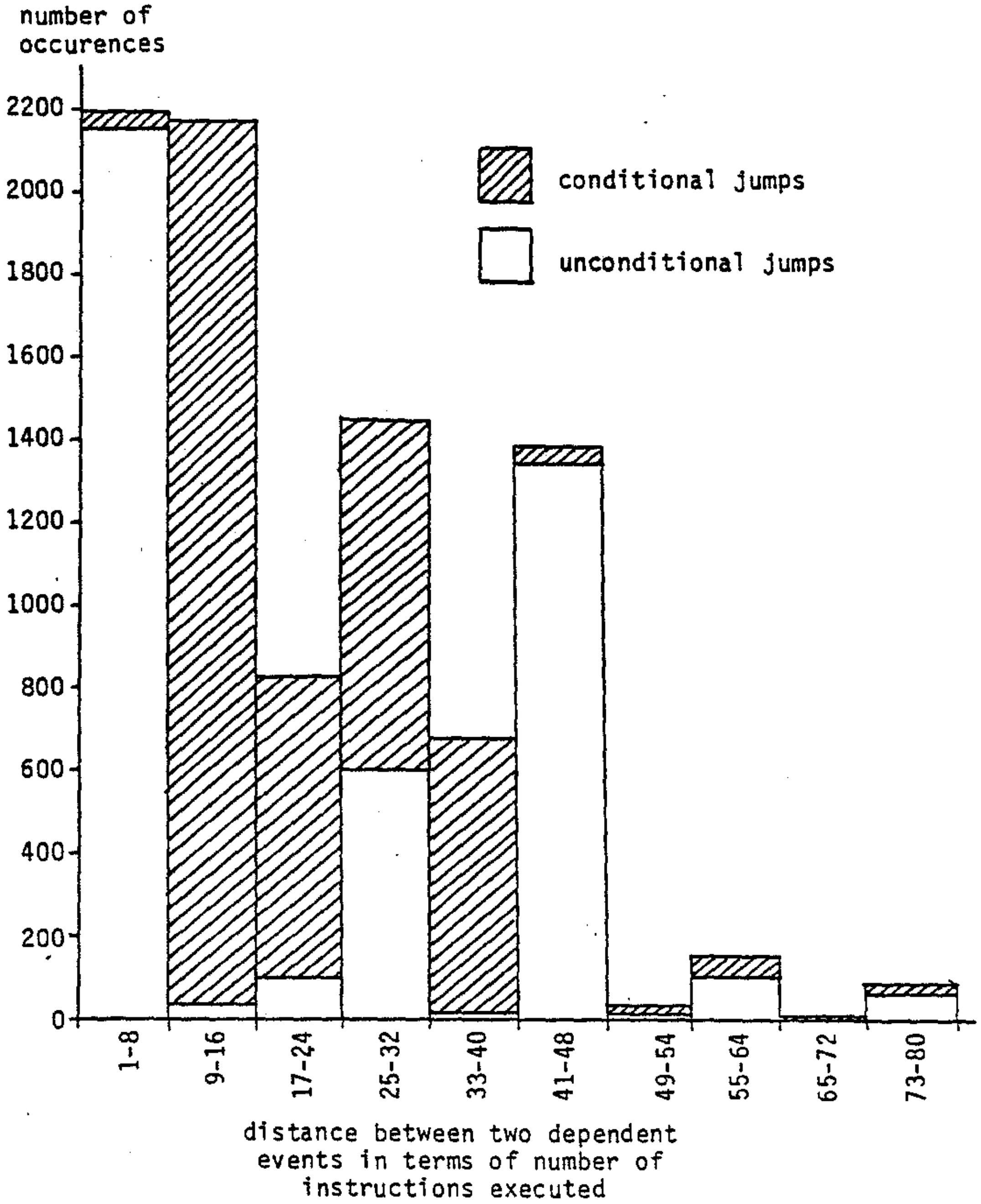


Figure 4.20 Histogram showing the Statistics of Dependent Events in Traces used

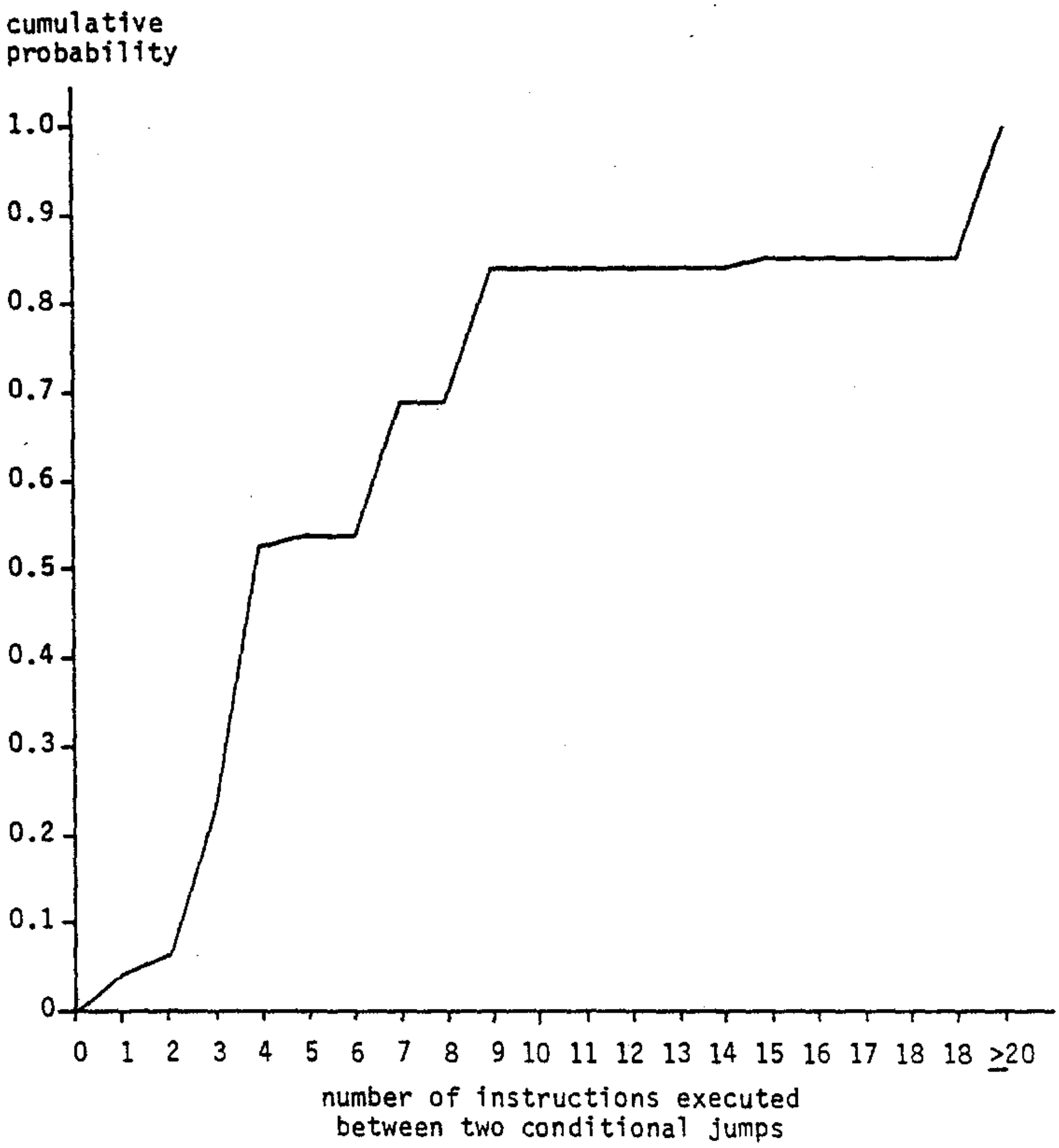
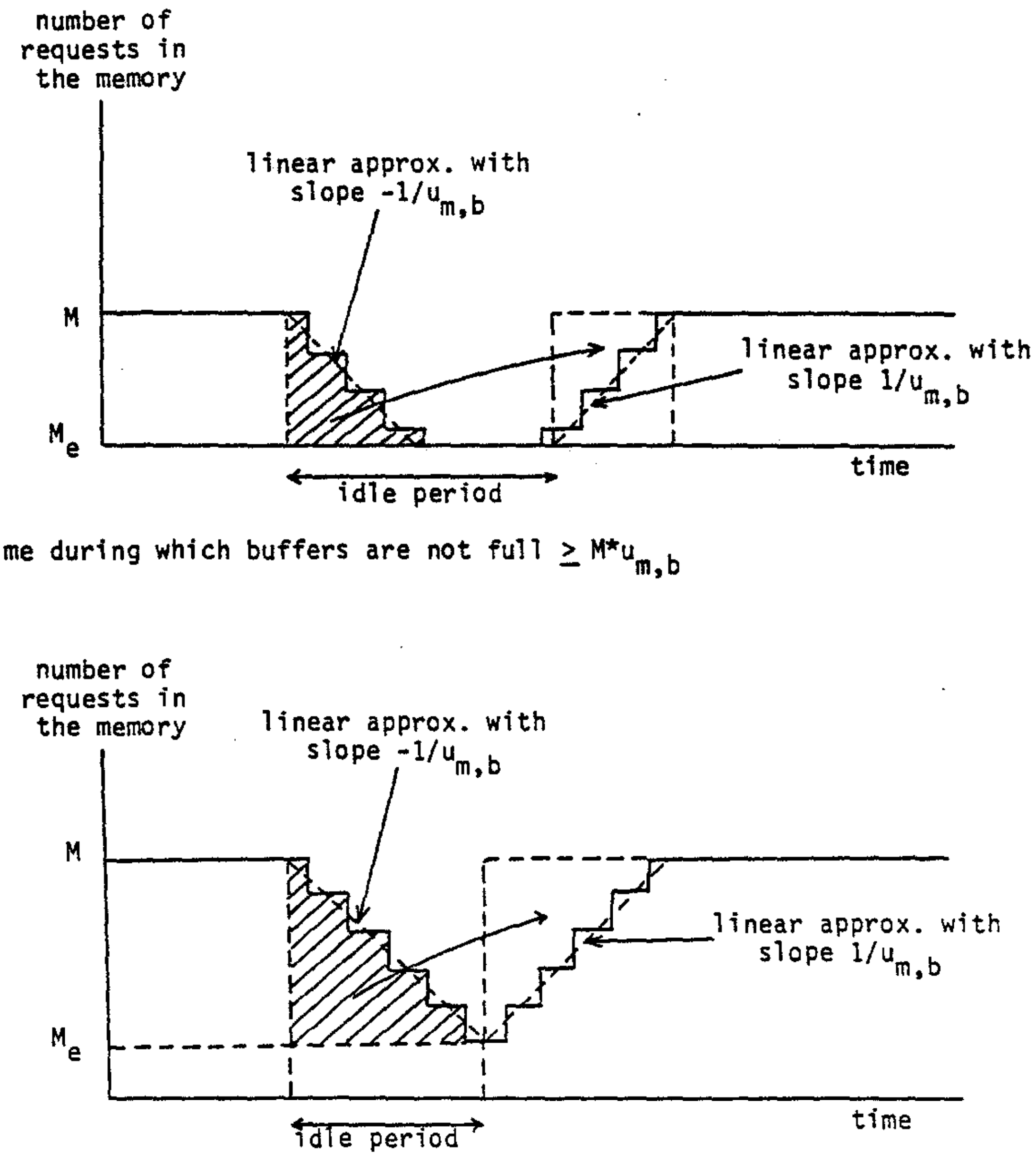


Figure 4.21 Graph showing Cumulative Probability of Number of Instructions executed between two Conditional Jumps


number of requests in the memory
linear approx. with slope $-1/u_{m,b}$
M
M_e
linear approx. with slope $1/u_{m,b}$
time
idle period
number of requests in the memory
linear approx. with slope $-1/u_{m,b}$
M
M_e
linear approx. with slope $1/u_{m,b}$
time
idle period

(a) Time during which buffers are not full $\geq M^*u_{m,b}$

(b) Time during which buffers are not full $\leq M^*u_{m,b}$

Figure 4.22 The Variation of the Number of Requests in the Buffers

jumps.

In the trace driven simulation results in Section 4.4.8.1, the instructions and its corresponding operands are assumed to be accessed one after the other. In the current model, the instructions are fetched much earlier than the corresponding operands. We have ignored these effects on the memory performance because there is very little correlation between the instruction address and its corresponding operand address (except in some cases, e.g. an architecture which implements the immediate mode, but the frequency of executing these instructions is small).

Since it is desired to find the maximum performance of the memory, the pipe must be designed in a way such that it is fast enough and long enough so that it is always able to fill up all the empty buffers in the memory within a memory sub-cycle. This design follows from our high request rate assumption. In this model, the pipe is essentially executing at the speed of the memory, that is, at rate $u_{m,b}/r$. The assumptions made are:

(1) There is a large amount of return buffers in the pipe for serviced requests. This assumption is necessary so that serviced requests can always be returned to the CPU without delaying the initiation of requests in the memory.

(2) Each segment in the pipe is very fast. This means that T is so small that if sufficient instructions are available to the decode unit, the pipe can generate enough requests to fill up all the memory buffers in one memory sub-cycle. This means:

$$T^*M \leq \frac{r}{u_{m,b}}$$

That is

$$T = \frac{r}{u_{m,b} * M} \tag{4.5}$$

(3) Since it takes a time W $(= w_{m,b} * m)$ to fetch an instruction, the pipe would have executed i*f instructions in this time interval at a rate of $\frac{u_{m,b}}{r}$ if no dependency occurs. Therefore

$$\frac{i*f}{u_{m,b}/r} > w_{m,b} * m$$

we set

$$f = \left\lceil \frac{w_{m,b} * u_{m,b} * m}{i*r} \right\rceil \tag{4.6}$$

where $\lceil y \rceil$ is the smallest integer larger than y. The value of f is chosen to be the smallest possible because when a conditional branch is encountered, one of the two paths is not traversed and therefore the instruction fetches for that path are wasted. The value of f is kept small in order to reduce the effects due to this waste.

(4) After an operand request is generated, the operand will be serviced after an average time W. In the meantime, the corresponding instruction passes through L-2 stages of the pipe in order to get to the execution unit. The time for this instruction to pass through the pipe must be longer than the waiting time for its operand so that the pipe is not blocked by this instruction waiting for its operand. We have

$$\frac{r*(L-2)}{u_{m,b}} \geqq w_{m,b} * m$$

We set

$$L = \left\lceil \frac{w_{m,b} * u_{m,b} * m}{r} \right\rceil + 2 \tag{4.7}$$

The value of L set in Eq. 4.7 is the minimum pipe length required for a maximum memory performance. For a longer pipe, the memory performance is lower because it takes a longer time for a dependent request to pass through the pipe. For a shorter pipe, the pipe is not able to generate requests fast enough because the last stage of the pipe is frequently blocked by unfinished operand requests. The value of L chosen is therefore a compromise between these two effects. These additional constraints can assure that the maximum performance of the memory is achieved.

When a conditional jump instruction is encountered and the condition code is set at a distance d stages away, the execution of the conditional jump is stopped until the instruction setting the condition code passes through L-d segments at a rate of $u_{m,b}/r$, if the conditional jump is unsuccessful. However, if it is successful, then the pipe is blocked until both the condition code is set and the target instruction has been fetched from the memory. If $t_e^{s/f}$ is set to be the time interval between the recognition of a successful/unsuccessful conditional jump and the time when the pipe can start execution again, then

$$t_e^s = \max\{\ (L-d)*\frac{r}{u_{m,b}},\ w_{m,b}*m\,\} \tag{4.8a}$$

$$t_e^f = (L-d)*\frac{r}{m} \tag{4.8b}$$

After the jump has been determined, it takes a small amount of time T/r to generate the operand request. It is not assumed that the decoding is done beforehand as in some machines. Let $t_r^{s/f}$ be the time interval from the recognition of a successful/unsuccessful conditional jump to the time when the pipe starts making requests. Then

$$t_r^{s/f} = t_e^{s/f} + \frac{T}{r} \tag{4.9}$$

After a dependent instruction has been encountered, there are still f instruction prefetch requests in the pipe. The idle period begins after these requests have been made to the memory. Let t_b be the time to make these remaining requests.

$$t_b = f * \frac{1}{u_{m,b}} \tag{4.10}$$

The length of the idle period ($ip^{s/f}$) is therefore the difference between t_r and t_b.

$$ip^{s/f} = \max\{\, 0,\ t_r^{s/f} - t_b \} \tag{4.11}$$

The above analysis is true for a particular value of d. Let D be the random variable denoting the distance, and D has the following distribution

$$Pr(D=d) = \begin{cases} p_d & d = 1, 2, \ldots, L \\ 0 & otherwise \end{cases} \tag{4.12}$$

This distribution is shown for the traces in Figure 4.21. Then combining Equations 4.8-4.12, we have

$$ip^s = \max \left\{ 0,\ \left[\sum_{d \geq L - \frac{w_{m,b} * u_{m,b} * m}{r}} (p_d * w_{m,b} * m) \right.\right. \tag{4.13a}$$

$$\left.\left. + \sum_{d < L - \frac{w_{m,b} * u_{m,b} * m}{r}} \left[p_d * \frac{(L-d) * r}{u_{m,b}} \right] \right] + \frac{T}{r} - t_b \right\}$$

$$ip^f = \max \left\{ 0,\ \sum_{d=1}^{L} p_d \frac{(L-d)}{u_{m,b}} + \frac{T}{r} - t_b \right\} \tag{4.13b}$$

Consider a time interval I, the number of accesses made during I is $u_{m,b} * I$ and the number of instructions executed in this interval is $\frac{u_{m,b} * I}{r}$. The average amount of idle period due to successful and unsuccessful conditional jumps are $x_{CJ}^{s} * ip^{s} * \frac{u_{m,b} * I}{r} + x_{CJ}^{f} * ip^{f} * \frac{u_{m,b} * I}{r}$. The resultant utilization is

$$u'_{m,b} = \frac{u_{m,b} * I}{I + x_{CJ}^{s} ip^{s} \frac{u_{m,b} * I}{r} + x_{CJ}^{f} ip^{f} \frac{u_{m,b} * I}{r}}$$

$$= \frac{u_{m,b}}{1 + \frac{u_{m,b}}{r} (x_{CJ}^{s} ip^{s} + x_{CJ}^{f} ip^{f})} \tag{4.14}$$

As a result of the degradation in memory utilization, there is a degradation in the buffer utilization. During an idle period, requests to the memory stop. At the end of the idle period, requests to the memory begin again. In Fig. 4.22, the decrease and the increase in the number of requests in the buffers are shown. Since M may not be an integer in our model (effective buffer length in Organization II), a linear approximation is used in the original function. In terms of the idle period, the time interval during which the buffers are not full is y = $ip^{s/f} + (M-M_e)*u_{m,b}$ where M_e is the effective number in the system. In fact, the shaded blocks in Fig. 4.22 can be rearranged so that the effective buffer utilization can be calculated. M_e, during an idle period in the two cases, is

$$M_e = \begin{cases} M - \dfrac{ip^{s/f}}{u_{m,b}} & \text{if } ip^{s/f} < M*u_{m,b} \\ 0 & \text{if } ip^{s/f} \geq M*u_{m,b} \end{cases} \tag{4.15}$$

Let $M_e^{s/f}$ be the effective number of requests in the system due to successful/unsuccessful conditional jumps and let M' be the resulting effective number in the system. For the time interval I,

$$\left\{\begin{matrix} total\ effective \\ buffer\ length \\ time\ product \end{matrix}\right\} = MI + M_e^s x_{CJ}^s \frac{u_{m,b} I}{r} ip^s + M_e^f x_{CJ}^f \frac{u_{m,b} I}{r} ip^f$$

That is

$$M' = \frac{M + \dfrac{u_{m,b}}{r}(M_e^s x_{CJ}^s + M_e^f x_{CJ}^f ip^f)}{1 + \dfrac{u_{m,b}}{r}(x_{CJ}^s ip^s + x_{CJ}^f ip^f)} \tag{4.16}$$

Using Little's Formula, the resulting number of waiting cycles $w'_{m,b}$ in the system can be calculated.

$$M' + u'_{m,b}*m = u'_{m,b}*w'_{m,b}*m$$

$$w'_{m,b} = 1 + \frac{M'}{u'_{m,b} m} \tag{4.17}$$

As a by-product of our estimation, it is not difficult to estimate the throughput of a memory bounded pipe. This will not be demonstrated here.

Using the statistics from the trace program, the results of the estimated utilization are plotted together with the simulation results in Figures 4.13 and 4.15. The degradation is quite significant and drops to about 50% of the original value in some cases. As seen in Fig. 4.13, the module utilization levels off much more rapidly with increasing buffer size than the original results with no dependencies. The curves plotted are not smooth because of the integrality requirement in the pipe length and the number of prefetched instructions . It is further seen that increasing buffer sizes do not improve the performance due to the effects of dependency. The difference in memory utilization for b=3 and b=10 is very small as seen in Fig. 4.15. The estimations for waiting cycles are not plotted in Figures 4.14 and 4.16 because they coincide almost exactly with the simulation results. In Fig. 4.23, the buffer utilization for Organization I with an MWFMF algorithm is plotted. It is seen that the buffer utilization is almost constant for large values of m. It is also interesting to note that the buffer utilization is lower for larger values of b. The explanation for this is because for a large value of b, the waiting time in the memory is longer and the memory utilization is higher. This implies that a longer pipe must be used (Eq. 4.7). A longer pipe means that it takes longer to resolve a dependent request and this causes degradation in the buffer utilization.

The above estimations only give an average value for the performance. In fact, if the memory can be utilized in some other way (e.g. for peripheral processing) when a dependency occurs, the degradation may not be so significant. The above analysis also reveals the fact that when the occurrences of dependent requests are frequent, it is not beneficial to use a pipelined computer in a batch mode. High degree of program interleaving using multiprogramming would help in reducing the degradation due to dependencies.

4.4.10 *Some final Remarks about the Design of Interleaved Memories*

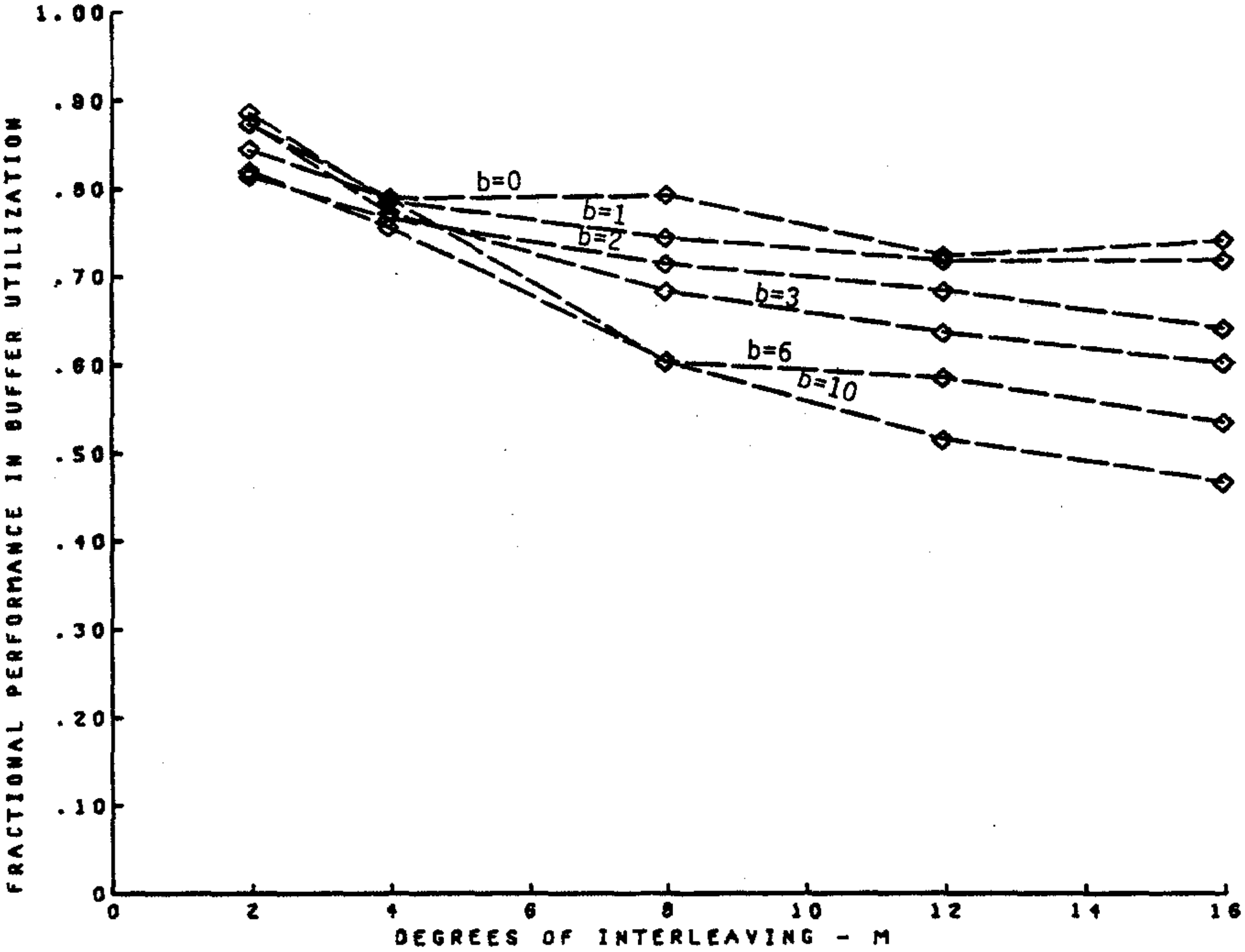


Figure 4.23 Buffer Utilization under Dependency for Org. I
with an MWFMF Scheduling Algorithm (Trace Driven Simulations)

We have presented in this section two organizations of an interleaved memory system which utilizes a finite buffer space for the storage of requests. We have designed a scheduling algorithm which allows a finite set of requests to be processed in the minimum expected time. However, the performance of our system is obviously less than the performance of systems with an infinite saturated request queue which is an unrealistic assumption. In Fig. 4.15, we have shown the performance of Hellerman's model [HEL67] together with our simulation results. Although Hellerman's model is a simple model and allows no queueing of requests, it is useful as a lower bound for the performance of other systems. It is seen that with a random request queue, Hellerman's model is better than our Organization I with $b = 0$, but is worse for $b > 0$. Note that the performance curves all have the same shape. Since Organization II degenerates into Organization I for $b = 0$, it is worse than Hellerman's model for $b = 0$, but better for $b > 0$. The comparison with other models in the current literature is not meaningful because they differ significantly.

We can improve our model slightly by considering the following. The rationale behind the constraint that only one module may be initiated in any sub-cycle is because the return bus can return at most one piece of datum in any sub-cycle. But since reads generate return data while writes do not, we can initiate two or more modules in a sub-cycle provided that exactly one of the requests is a read. The improvement in utilization due to this is only about 2%. The improvement is not significant because the fraction of writes in our trace is less than 7% of all the accesses and its applicability is also limited by memory interference.

The questions that still remain to be resolved are how can one select between Organization I and Organization II and how does one choose the parameters of the system in order to satisfy all the requirements. In the hardware requirements, Organization I needs associative search capabilities in the buffers while Organization II does not. However, the availability of fast associative memory, (see chapter 5 of this thesis), can help in this regard. The performance of Organization II predicted may be worse because it may require the transfer of more than one request into the memory system during a memory sub-cycle and it sometimes is not possible in a pipelined system. Organization II gives a slightly worse performance than Organization I when a maximum of one request is allowed to be generated in each sub-cycle and the effective buffer sizes in both organizations are identical. Tradeoff in cost and performance must be made in the selection of the organization. In order to answer the second question we have raised, we need to design a cost model of the system. The cost of individual component is highly technology dependent and will not be discussed here. However, the designer can find a configuration with the minimum cost based on the bandwidth and the response time requirements. Assuming that the bus width is determined and fixed, he can use the average utilization (a function of the degrees of interleaving) as an alternate measure of bandwidth. The response time can also be normalized with respect to the speed of the memory to give the waiting cycle. In the above calculations, the effects of dependency are not considered, otherwise, Equations 4.14 and 4.17 can be used to find the values of utilization and waiting cycle with dependency. Using Little's Formula, the average number of requests in the memory, or the average number in the request buffers can be obtained. The designer can then substitute the values for the average utilization and the buffer size into the formula obtained by regression (Table 4.3) to get a polynomial equation as a function of the degrees of interleaving and the memory speed. By evaluating the speed for different possible degrees of interleaving, the cost of the memory can be estimated. The final configuration selected will be the one with the minimum cost.

The MWFMF scheduling algorithm we have studied in this section is optimal in the sense that it minimizes the expected finish time for a finite sequence of random, independent requests. Although there exists restrictions and the performance of specially structured computers, e.g. CRAY I, ILLIAC IV, etc are not found, our scheduling algorithm is applicable to machines which support vector-oriented computing, e.g. TIASC, and array type processors like ILLIAC IV.

The organizations we have presented in this section can be extended to other levels of the memory hierarchy in which the modules can be disks and the requests can be disk requests instead of memory addresses. The service time distribution of a disk is not constant as in the case of a memory module. However, some approximation can be made on the distribution (e.g. by an exponential distribution) and analysis techniques in queueing theory can be applied to the model [BAS75]. In the next section, we return to the original task scheduling problem on the general model. We show a heuristic to schedule tasks and the heuristic is evaluated by simulations.

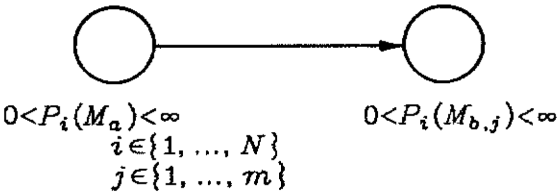
4.5 *A HEUURISTIC FOR THE SCHEDULING OF TASKS ON THE GENERAL MODEL*

We have presented in detail in the last section the design of an interleaved memory which is a restriction of the general model we have described in Section 4.2. Although the task scheduling problem on the general model is NP-complete, we see that an optimal average behavior algorithm can be designed when the model is sufficiently restricted.

We would like to return to the original task scheduling problem on the general model. Since the problem is NP-complete, a heuristic should be designed if it is not feasible to enumerate over all the possibilities in order to find the optimal sequence. We present in this section a heuristic for the task scheduling problem on the general model. This heuristic is extended from Johnson's optimal two stage flow shop algorithm [JOH54], and the performance of the heuristic is seen to perform reasonably well in a limited number of simulations.

The heuristic is designed for tasks with the following characteristics:

(1) Each request has the following precedence graph:



(2) There are no precedence constraints among requests.

(3) No preemption is allowed.

(4) $buff\,(M_a) = \infty$ and $r_i = 0$ ($i \in \{1, \ldots, N\}$).

This says that the release times of jobs are 0 and the buffer size of M_a is very large, that is, all the requests are available initially for scheduling.

(5) $0 < buff\,(M_{b,1}) = \cdots = buff\,(M_{b,m}) < \infty$

That is, all the modules in the second stage have finite, non-zero amount of buffers.

(6) Permutation schedule is desired.

The heuristic for scheduling this class of jobs is:

Algorithm 4.4: Heuristic to Schedule Tasks on the General Model

1. Order jobs that require the service of $M_{b,j}$ ($j=1,\ldots,m$) in increasing ratios of:

   $P_i(M_{i,j)}-p^*P_i(M_a)$

2. Merge the job sequences for different $M_{b,j}$'s into one stream using the MWFMF scheduling algorithm (Algorithm 4.3).

3. In the schedule obtained, for any continuous sequence of jobs that require the same module on the second stage, reorder using Johnson's Algorithm such that x should be scheduled before y if

$$\min \{P_x(M_a), P_y(M_{b,j})\} \leq \min \{P_x(M_{b,j}), P_y(M_a)\}$$

In step 1 of the algorithm, p is a constant to be selected. The rationale behind why the jobs have to be ordered in this fashion is because it is better to schedule jobs with smaller processing requirements on M_a first. By doing this, the processing on the second stage can be started earlier than if a job with a large processing requirement is started on M_a first. Even if two sequences finish at the save time, there is more leverage for adjustment in a sequence which starts the processing on the second stage earlier. This step only orders jobs for each module on the second stage. Step 2 of the algorithm merges these sequences together. Since the MWFMF algorithm (Algorithm 4.3) is found to perform very well, it is also applied here. Lastly, the sequence obtained can still be improved if any two consecutive jobs in the sequence which require the service of the same module on the second stage are rearranged using Johnson's algorithm [JOH54]. The reason is because Johnson's algorithm minimizes the finish time of a sequence on a two stage flow shop and we are treating M_a and one of the $M_{b,j}$'s as a two stage flow shop in this consideration. Some simulations were done to determine the performance of this heuristic. This is shown in Figure 4.24. The results are plotted for 1000 samples of 7 randomly generated jobs. It is assumed that $m=2$, $p=2$ and four of these jobs require the service of $m_{b,1}$ and three require the service of $M_{b,2}$. Although the amount of simulations is limited, it is seen that the performance of the heuristic is very good. Approximately 67% of the simulations have no deviation from the optimal performance and only about 1% of the simulation deviate by 28% from the optimal performance. The exact worst case and average case performance are difficult to be derived analytically.

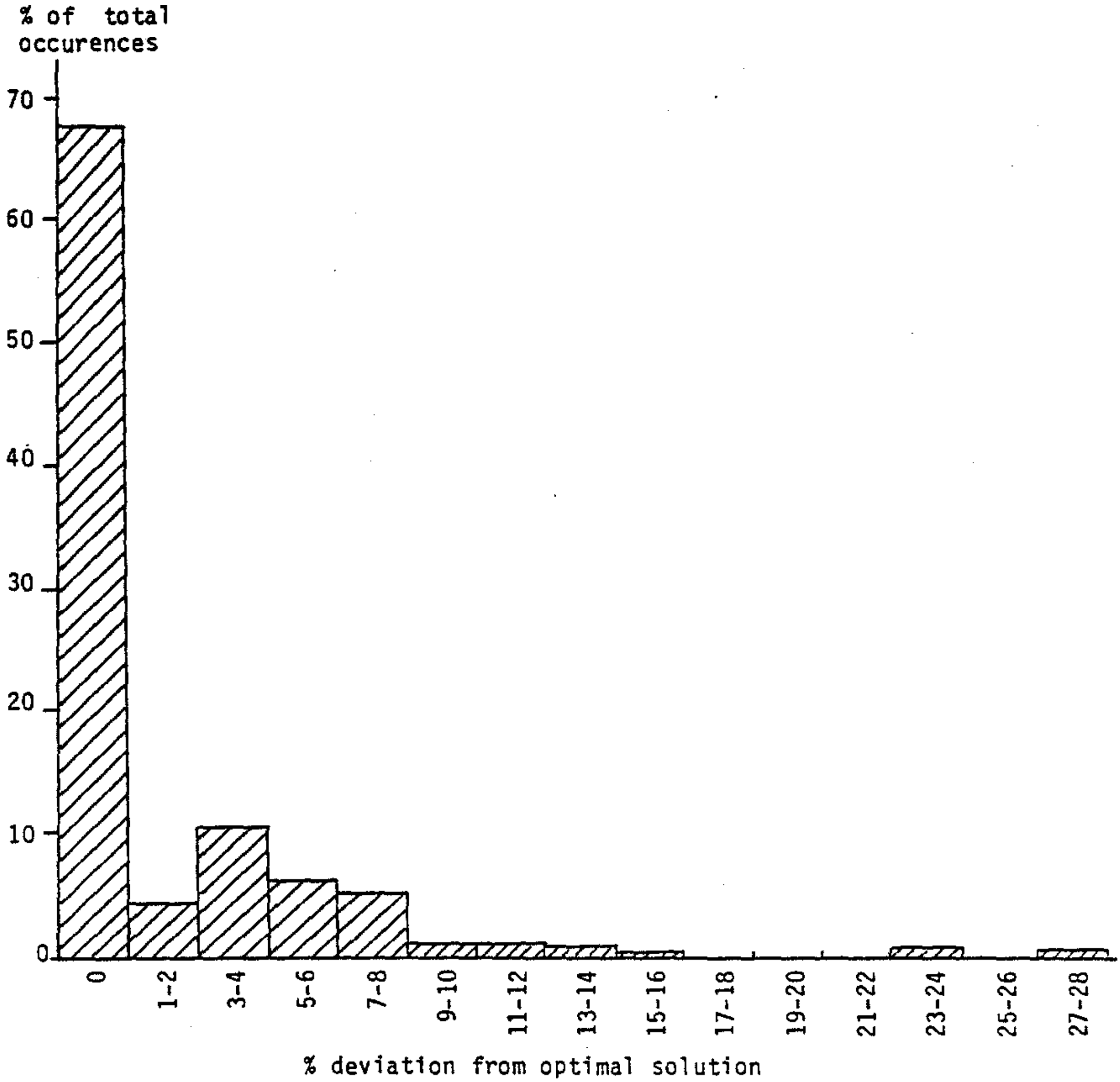


Figure 4.24 Simulation Results for Algorithm 4.4 using 1000 Samples of 7 Randomly Generated Jobs (m=2, p=2)

4.6 *CONCLUSION*

In this chapter, we have studied the task scheduling problem on a DCS. This problem is related to the scheduling of tasks after the query has been decomposed and the files have been placed on the DCS, and is more related to the hardware architecture of the system. Because it is difficult to collect global information on the DCS, most of the scheduling decisions have to be made locally. We have therefore restricted the general task scheduling problem to the problem of scheduling tasks at each node independently. The model for such a system is the SIMD model proposed by Flynn [FLY66].

The contributions that we have made in this chapter are:

(1) We have proved the NP-completeness of the task scheduling problem on the SIMD model. These include the cases when the jobs have positive release dates, precedence constraints or no waiting space in the second stage. Therefore it is unlikely that an optimal sequence can be obtained without exhaustive enumeration.

(2) We have put additional constraints on the model so that the problem becomes polynomially solvable. We restrict the processing times of jobs so that they are constant and the ratio of processing times of the second stage to the first stage is m (where m is the number of modules on the second stage). We further assume that each job requires the service of one of the modules on the second stage. The resulting model is a model of an interleaved memory system for a pipelined processor. We have evaluated several alternative scheduling algorithms and have proven that one of these algorithms minimizes the expected completion time for a finite set of random requests. This algorithm is therefore an optimal average behavior algorithm. We have also evaluated the degradation in performance due to dependencies in the access stream.

(3) We have designed a heuristic for task scheduling on the general model. This heuristic is extended from Johnson's two stage flow shop algorithm [JOH54]. Although the algorithm is evaluated with a limited number of simulations, the performance is seen to be very good.

In the next chapter, we study some hardware support aspects for data management on a DCS. One particular hardware necessary is the associative memory which we have used in the design of interleaved memories (Section 4.4). This associative memory must be capable of performing equality, and maximum searches. There are other hardware designs which are needed to support data base operations such as simple retrievals and updates, threshold, proximity and minimum searches. We make use of the current LSI technologies to design some supporting hardware for data management.

5. *HARDWARE SUPPORT FOR DATA MANAGEMENT ON DISTRIBUTED COMPUTER SYSTEMS*

5.1 *INTRODUCTION*

In the past three chapters, we have discussed some logical solutions to data management on a DCS. Some of these solutions do not require specialized hardware support, e.g. query decomposition, file placement and migration, while some others require dedicated hardware, e.g. the associative memory used in request scheduling on an interleaved memory. In general, there is a tendency for increasing hardware support for data management functions on a DCS. The motivations for this tendency of functional distribution are:

(1) *Parallelism*

As the size of information processing grows, it becomes increasingly difficult to use a uni-processor to achieve the system's requirements. One alternative is to exploit the possibility of using multiple, less expensive and less powerful processors to form a conglomerate of parallel processors which can usually achieve the system's requirements in a more cost-effective way.

(2) *Communication overhead*

Processing on large file systems are often I/O bound. Many of the file operations are quite simple and a significant communication overhead is incurred in transferring the file to a level of the memory hierarchy where the processor can process it. By distributing the intelligence to the different levels of the memory hierarchy, parallel processing can be performed with very little communication overhead.

(3) *Hardware or firmware realization of data base functions*

The complexity of data base system software is largely due to the processing of memory mapping operations. Memory mapping operations convert the file accesses by a query into actual memory addresses and must be highly optimized if they are to perform well. These operations often utilize complex data structures to achieve efficiency. On the other hand, data base software are divided into modules which perform specific tasks. For example, modules may exist for query parsing, directory access, directory processing, data retrieval and update, and data security. These modules usually have diverse capabilities, and bottlenecks exist if these modules are executed on the same processor. The system performance is consequently degraded. Specialized data base hardware solves the above two problems by eliminating the complex address mapping operations and utilizing hardware/firmware to replace the software. The query is transferred directly from the processor to the specialized hardware without address mapping. Then hardware/firmware will process the query and realize the data base functions.

As a result, there are many haredware designs proposed to speed up data base processing. One of the earliest design is the associative memory [SLA56] in which logic is distributed into each cell of the memory so that search operations can be performed associatively. Such a design is rather expensive because logic is duplicated for each bit in the memory. A more recent design is the data base machine [HSI77] which is a remedy to the costly associative memory by sharing one piece of the associative logic among a set of physically related data. A set of physically related data may be a track on a disk in which case the design is a logic per track disk; it may be a set of memory modules, in which case the design is an SIMD computer model [FLY66]. Since the designer has the freedom of designing the degree and the amount of parallelism, there are a lot of issues related to the data base machine design. Some of these issues will be addressed in Section 5.3.

In this chapter, we present a design of an associative memory for ordered retrieval in Section 5.2 and extend the design to a simple data base machine in Section 5.3. In the associative memory design, we present some simple schemes for a variety of searches, each of which may be performed in one complete memory cycle using bit-memory logic primarily. The searches we study include the basic equality search, the threshold searches (both greater than and less than searches), the proximity search, and most importantly, the greatest value and the least value searches. For each kind of search, we present both the algorithm suitable for our needs and the logic circuit of the memory cell required by the algorithm. Based on the basic search schemes, an algorithm for ordered retrieval is developed. A comparison for ordered retrieval schemes is then made between the proposed scheme and the previous algorithms. It is found that this algorithm outperforms all the other algorithms compared, particularly in the resolution of multiple responses. Finally, issues relating to LSI implementation, manufacturing defects, and modular expansions are discussed.

In the data base machine design, we investigate some problems that exist with the design and show that the design should be made as a combination of SIMD and MIMD computer models [FLY66]. Lastly, we show the extension of the associative memory to the design of a simple data base machine.

5.2 *A DESIGN OF A FAST CELLULAR ASSOCIATIVE MEMORY FOR ORDERED RETRIEVAL*

5.2.1 Previous work

Content-addressable memories (CAM's), alternatively known as associative memories (AM's), have received much attention in the literature since they were first described in 1956 [SLA56]. The distinguishing feature of such memories is that stored words are accessed by matching some portion of their contents to a search word and selecting the first one that matches rather than accessing the data using its physical location in the memory as in standard random access memories (RAM's). It can be readily seen that CAM's must depend upon a high degree of parallelism in their search schemes in order to compete in memory access times with RAM's. Large speed improvements can be gained from this parallelism and this makes CAM's attractive to a wide variety of applications. A good survey of the current technology in CAM's can be found in [PAR73, FOS76, HAN66].

With the advent of large scale integration (LSI) technology, it becomes feasible to economically implement fast search algorithms in CAM's by incorporating much of the control logic into the memory plane. Several search algorithms for CAM's have been developed in the past two decades [FEN74, FRE61, SEE62, LEW62]. Some algorithms, such as [SEE62] and [YAN66] have been based upon distributed logic design, but few have incorporated a high percentage of their search logic in the memory cell. An exception to this is found in a design by Kautz [KAU69] for a special purpose sorting array. His design is oriented towards ordering, rather than searching, of the memory, but does include associative capabilities as a byproduct.

The trend in associative memory design is toward distributed logic. Previous designs have placed control logic outside the storage logic. These control logic include comparison logic, propagation logic, multiple response resolution logic, arithmetic logic, etc. In a distributed logic design, the control logic and the storage logic are designed together. The controls are brought into the cells as part of the storage itself. The cells become more complex and have more control functions associated with them, but it also results in more homogeneous and modular design. In this section, we propose the basic design of such a memory and present some searching and sorting schemes and the implementation of some basic searches using distributed cellular logic which is considerably faster than any of the previous sorting methods. The capabilities of the cells are actually a subset of the capabilities of Kautz's augmented CAM array [KAU71]. Further, the concept and the design of some of the searches havn been investigated earlier [TUR72, RAM78a]. The searches that we examine include the basic equality search, the threshold searches (both greater than and less than searches), the proximity search, and most importantly, the greatest value search and the least value search.

5.2.2 *Symbols used in the Design*

The following conventions are used throughout the design:

B_i The value of the i'th word of memory;

$B_{i,j}$ the value of the j'th bit of the i'th word of memory;

C a priority circuit which is used to sequence response in W_2, W_3, W_4 *or* W_5;

D a circuit used to detect responses in W_2, W_3, W_4 *or* W_5;

$e_{i,j}$ the equality state signal for the j'th bit of the i'th word in the equality match between $B_{i,j}$ and R_j;

$E_{i,j}$ the equality enable signal for the j'th bit of the i'th word in the equality-inequality search mode and the least value search mode;

$E_{i,n+1}$ signal which can be gated to set (or reset) any one of the word control registers W_2, W_3, W_4 *or* W_5;

$F_{i,j}$ the enable signal for the j'th bit of the i'th word in the greatest value search;

$F_{i,n+1}$ signal which can be gated to set (or reset) any one of the word control registers W_2, W_3, W_4 *or* W_5;

G the associative memory search mode command (equality-inequality-proximity mode or the least value mode);

i an index for a word in the memory, $1 \leqq i \leqq m$;

I_j the value of the j'th bit of the input/output register I;

j an index for a bit in the word, $1 \leqq j \leqq n$;

k a variable index, $1 \leqq k \leqq n$;

L_i the less than state signal for the i'th word of memory; a signal which can be gated to set (or reset) any one of the word control registers W_2, W_3, W_4 *or* W_5;

LSB Least Significant Bit;

m the number of words in the CAM;

M_j the value of the j'th bit of the mask register M, (used in the least value search, the equality search, the threshold searches and the proximity search);

M_j^* the value of the j'th bit of the mask register M^* (used in the greatest value search);

MSB Most Significant Bit;

MZ the set of all bit positions with $M_j = 0$;

n the number of bits in a word of the CAM;

P_j the synchronization bus signal for the j'th bit-slice in the least value search;

Q_j the default-detection bus signal for the j'th bit-slice in the least value search;

r an index in the word control logic, $1 \leqq r \leqq 5$;

R_j the signal for the j'th bit-slice shared by the equality-inequality search, the proximity search and the least value search;

S the value of the search register S;

S_j the value of the j'th bit of the search register S;

T_j the search-default feedback bus signal for the j'th bit-slice in the greatest value search;

U_j the synchronization bus signal for the j'th bit-slice in the greatest value search;

V_j the default-detection bus signal for the j'th bit-slice in the greatest value search;

$w_{i,r}$ the i'th flip-flop of the word control register W_r;

W_1 the word flags register with m flip-flops;

$W_2 - W_5$ result stores or temporary stores in the word control logic;

$X_{i,j}$ the proximity state signal for the i'th word of the memory in the proximity search;

$X_{i,n+1}$ a signal which can be gated to set (or reset) any of the word control regisers W_2, W_3, W_4 or W_5;

$\cup$ abbreviation for logical OR operation;

$\in$ abbreviation for "an element of";

$\forall$ abbreviation for "for all";

$\exists$ abbreviation for "there exists".

5.2.3 *Basic Associative Memory Organization*

The associative memory organization shown in Fig. 5.1 is used to implement the search schemes to be presented. A *bit- slice* is a vertical slice through the memory as arranged in Fig. 5.1. The j'th bit-slice is made up of the j'th bit of

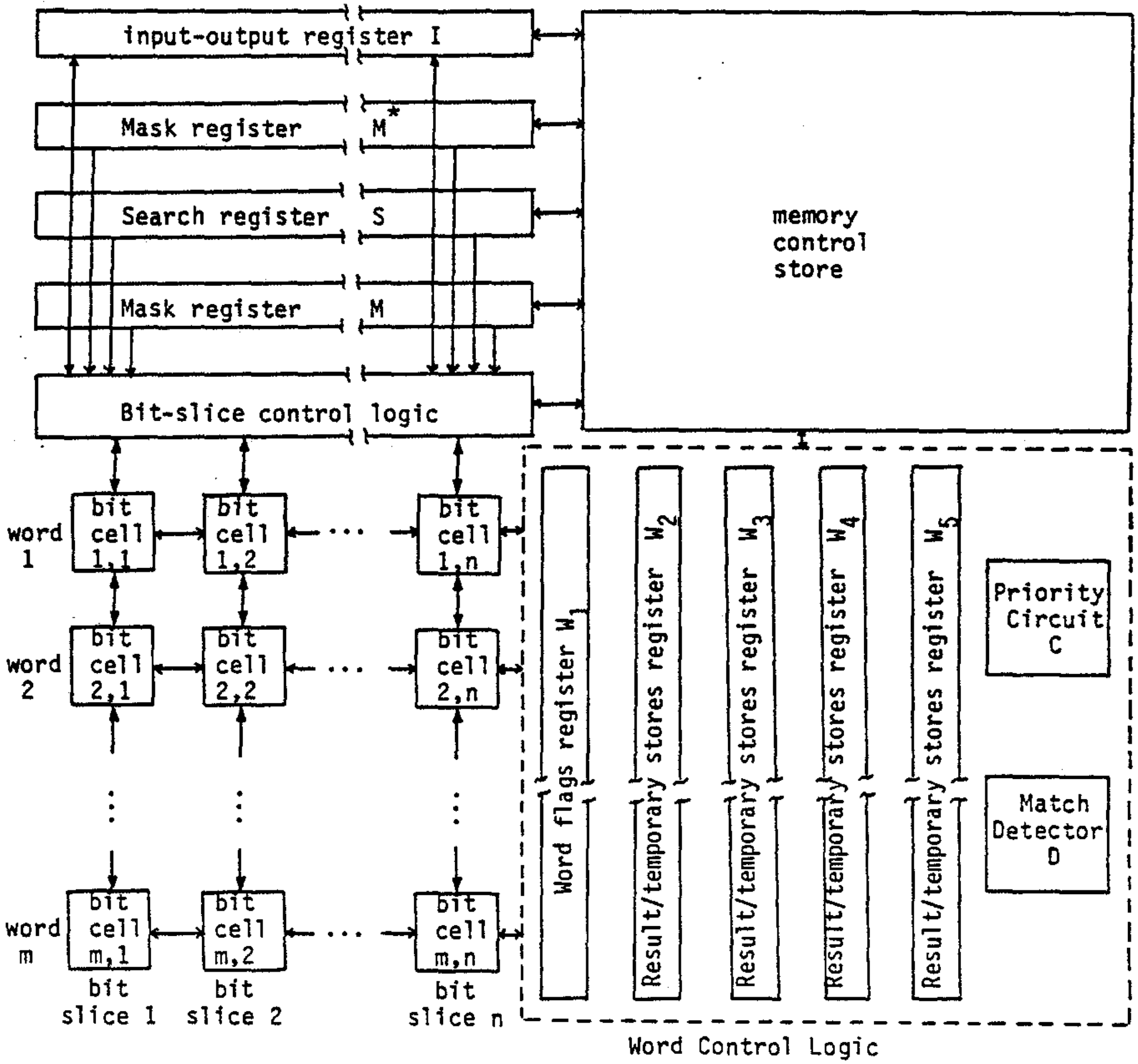


Figure 5.1 Cellular Logic Associative Memory Block Diagram

every word in the memory. The search operations are parallel by word and serial by bit-slice. A *minor cycle* refers to the time needed to perform an operation on a single bit-slice and a major cycle refers to the time needed to complete an operation on all bit-slices of the memory. Hence, a *major cycle* for the present AM organization is composed of n minor cycles where n is the number of bits in a word. It is shown later that some searches will require a longer minor cycle than others, thereby lengthening the major cycle as well. A *"basic" operation* is an operation which may be performed in a single major cycle.

5.2.4 *Definition of Search Operations*

In each of the following search definitions, the set of words involved in the search are those where $w_{i,1}=1$ and $i \in \{1, 2, ..., m\}$. The result of the search partitions this set of words into two sets, the set that satisfies the search condition and the set that does not. Let B_i be the content of the i'th word in the memory, S be the content of the search register, and M be the content of the mask register. That is,

$$B_i = \sum_{j=1}^{n} 2^{n-j} \cdot B_{i,j},$$

$$S = \sum_{j=1}^{n} 2^{n-j} \cdot S_j$$

and

$$M = \sum_{j=1}^{n} 2^{n-j} \cdot M_j.$$

The search is performed only on that part of the search word which is not masked. In other words, only those S_j bits for which the corresponding M_j bits are 0's are included in the matching (comparison) process. Let MZ be this set of bit positions. Other bit positions with $M_j=1$ are bypassed. (Note that j=1 for MSB and j = n for LSB.) We define the various searches as follows:

A. *Equivalence Searches*

1) *Equality Search:*

$$B_{i,j}=S_j, \forall j\in\{1, 2, \ldots, n\}.$$

2) *Inequality Search:* $\exists| k \in MZ$ *such that* $B_{i,k}\neq S_k$.

3) *Similarity Search (Masked Equality Search):* $B_{i,j}=S_j \ \forall j \in MZ$.

4) *Proximity Search:* There is exactly one k ∈ MZ such that $B_{i,k}\neq S_k$.

Note: The similarity search is also known as masked-equality search. It differs from the equality search in that the mask is effectively not used in the latter while it is used in the former search. In most cases, this distinction is so insignificant that the "equality search" is used to mean both the equality and the masked-equality searches. Unless specified otherwise, we will assume that all searches are masked.

B. *Threshold Searches*

1) *Greater- Than Search:* $B_i>S$.

2) *Less- Than Search:* $B_i<S$.

3) *Greater- Than- or- Equal- To Search:* $B_i\geqq S$.

4) *Less- Than- or- Equal- To Search:* $B_i\leqq S$.

C. *Double- Limits Searches*

1) *Between- Limits Searches:* Let X and Y be the limits such that X > Y. Then B_i is

a) < X and > Y,

b) < X and ≧ Y,

c) ≦ X and > Y,

d) ≦X and ≧ Y.

2) *Outside- Limits Searches:* Let X and Y be the limits such that X < Y. Then B_i is

a) < X of > Y,

b) $< X$ or $\geq Y$,

c) $\leq X$ or $> Y$,

d) $\leq X$ or $\geq Y$.

D. *Extremum Searches*

1) *Least Value Search:* $B_i \leq B_k$, $\forall k \neq i$ and $k \in \{1, 2, ..., m\}$.

2) *Greatest Value Search:* $B_i \geq B_k$, $\forall k \neq i$ and $k \in \{1, 2, ..., m\}$.

E. *Adjacency Searches*

1) *Nearest- Above Search:* $\nexists k \in \{1, 2, ..., m\}$ such that $B_i > B_k > S$.

2) *Nearest- Below Search:* $\nexists k \in \{1, 2, ..., m\}$ such that $B_i < B_k < S$.

There are other non-search operations that can be performed in associative memories. These include word addition, field addition, summation, counting, shifting, complementing, logical sum, logical produuct, etc. Devices that incorporate non-search operations may be referred to as associative processors [FOS76]. We will not investigate further on non-search operations in this chapter.

5.2.5 *Algorithms and Implementations of Basic Searches*

We define a *basic search* as one which can be completed in exactly one major cycle, assuming multiple response resolution as an operation separate from search operations. This definition applies only to the configuration of the CAM in Fig. 5.1. A *multiple response* is a situation when more than one word satisfies the given search condition. The multiple response resolution resoles this situation by means of a priority circuit [FOS68] or other schemes, e.g., [LAN77, HIL66a, WEI63], and outputs all the responders one at a time. Among the searches listed in the previous section, not all of them can be economically implemented as basic searches. Therefore, we choose to implement those searches which are most frequently used as basic searches while the rest can be performed in a series of the basic searches. As an example, the between-the-limits search (Y $\leq B_i < X$) can be generated by performing a less-than search (<X) followed by a greater-than-and-equal-to search ($\geq Y$) on the reponders of the first search. In the implementation to be presented, the basic searches are the equality search, the similarity (masked equality) search, the proximity search, the four threshold searches, and the two extremum searches. Each of these basic searches can be performed alone or a few combinations of them can be performed simultaneously. These searches are grouped into three groups called Mode A, Mode B, and Mode C operations. The Mode groupings are as follows.

Mode A: The equality search, the similarity search, the proximity search and the four threshold searches.

Mode B: The least value search.

Mode C: The greatest value search.

Searches in Mode A can be performed simultaneously. Furthermore, Mode A or Mode B operations can be performed simultaneously with Mode C operations. We will assume that positive logic is used throughout our designs. It should be noted that not all of these searches are required in a specific application. They are presented here for completeness.

5.2.5.1 *Mode A: Equality- Threshold- Proximity Search Mode*

In the equality-threshold search, the CAM is partitioned according to the magnitude of the search word S into three sets, namely, words which are equal to S, words which are less than S, and words which are greater than S. The result of this search mode is stored in two of the word control registers, W_2 and W_3, and the interpretation is given in the algorithm to follow. Further, in the proximity search mode, the CAM is partitioned into two sets, words which are near to S, and words which are not. The results of this are gated into W_4. Note that if it is not necessary to preform the proximity search together with the threshold searches, then register W_5 can be eliminated from the design. This search mode is characterized by the signal $G=0$, which gates the contents of the search register S to the search bus. That is, $R_j = S_j \forall j \in \{1, 2, ..., n\}$. The basic searches performed in this mode are the equality search, the four threshold searches (namely, >S, <S, $\geq S$, and $\leq S$), and the proximity search. $M_j=0$ means that S_j is not masked while $M_j=1$ means S_j is masked.

The three query states are shown in the following table.

M_j	S_j	Query State
0	0	0
0	1	1
1	0	d
1	1	d

d = don't care

Algorithm 5.1 - Mode A Search Operation

```
1) $\forall i \in \{1, 2, ...., m\}$
   a) Initialization:
      S ← Search word, M ← Mask, G = 0, j = 0,
      $w_{i,1} = 1$, $w_{i,2} = 0$, $w_{i,3} = 0$, $w_{i,4}=0$.
   b) Data Path Setting:
      Gate $X_{i,n+1}$ to $W_{i,4}$, $E_{i,n+1}$ to $w_{i,3}$, $L_i$ to $w_{i,2}$, $w_{i,1}$ to $E_{i,1}$, $\overline{W}_{i,1}$ to $X_{i,1}$.
      These data paths and the control signal G are held until the completion of the major cycle.
2) Let j ← j + 1.
3) Compute $\forall i \in \{1, 2, ..., m\}$ simultaneously
   a) $e_{i,j} = (M_j + B_{i,j} \cdot R_j + \overline{B}_{i,j} \cdot \overline{R}_j)$
   b) $E_{i,j+1} = E_{i,j} \cdot e_{i,j}$,
   c) $d_{i,j} = E_{i,j} \cdot \overline{B}_{i,j} \cdot \overline{e}_{i,j}$, [1]
```

[1] $d_{i,j}$ will be sensitive to $\overline{B}_{i,j}$ and only to the first bit mismatch between B_i and S. A simpler

d)

$$L_i = \bigcup_{k=1}^{j} d_{i,k} \text{(wired-OR)},$$

e) $X_{i,j+1} = X_{i,j} \cdot e_{i,j} + \overline{X}_{i,j} \cdot \overline{e}_{i,j} \cdot E_{i,j}$

4) *Is $j = n$?*

a) Yes - Proceed to step 5).

b) No - Proceed to step 2).

5) *Result Interpretation:*

For those words i with $W_{i,1}$=1,

$w_{i,2}(=L_i)$	$w_{i,3}(=E_{i,n+1})$	Interpretation (Equality-Threshold search)
0	0	B_i is greater than search word.
0	1	B_i matches search word
1	0	B_i is less than search word
1	1	[does not occur]

$W_{i,4}\ (=X_{i,n+1})$	$W_{i,3}\ (=E_{i,n+1})$	Interpretation (Proximity search)
0	0	B_i is not near to search word
0	1	B_i matches search word
1	0	B_i is near to search word
1	1	[does not occur]

In this search algorithm, the minor cycle is composed of step 3) alone while the major cycle is composed of steps 2)-4). The result of this search mode is handled by the match detector D in the word control logic. Any multiple responses is resolved by the priority circuit C. The bit-cell logic needed to implement this equality-threshold-proximity search mode is shown in Fig. 5.2. The delay in each minor cycle is one gate delay for the equality-threshold searches and three gate delays for the proximity search. The following example shows the state of L_i, $E_{i,j+1}$ and $X_{i,j+1}$ for a Mode A search of 6 words, each 5 bits

design using $d_{i,j}=\overline{E}_{i,j+1} \cdot \overline{B}_{i,j}$ can be used. In this case, $d_{i,j}$ will be sensitive to all mismatches between B_i and S. Since L_i is obtained by wired-ORing $d_{i,j}$'s, the final output voltage of the wired-OR will depend on the number of mismatches. It will be more appropriate to eliminate this dependence by only taking the first mismatch as what is done here. We must confess that the exact design is highly technology-dependent.

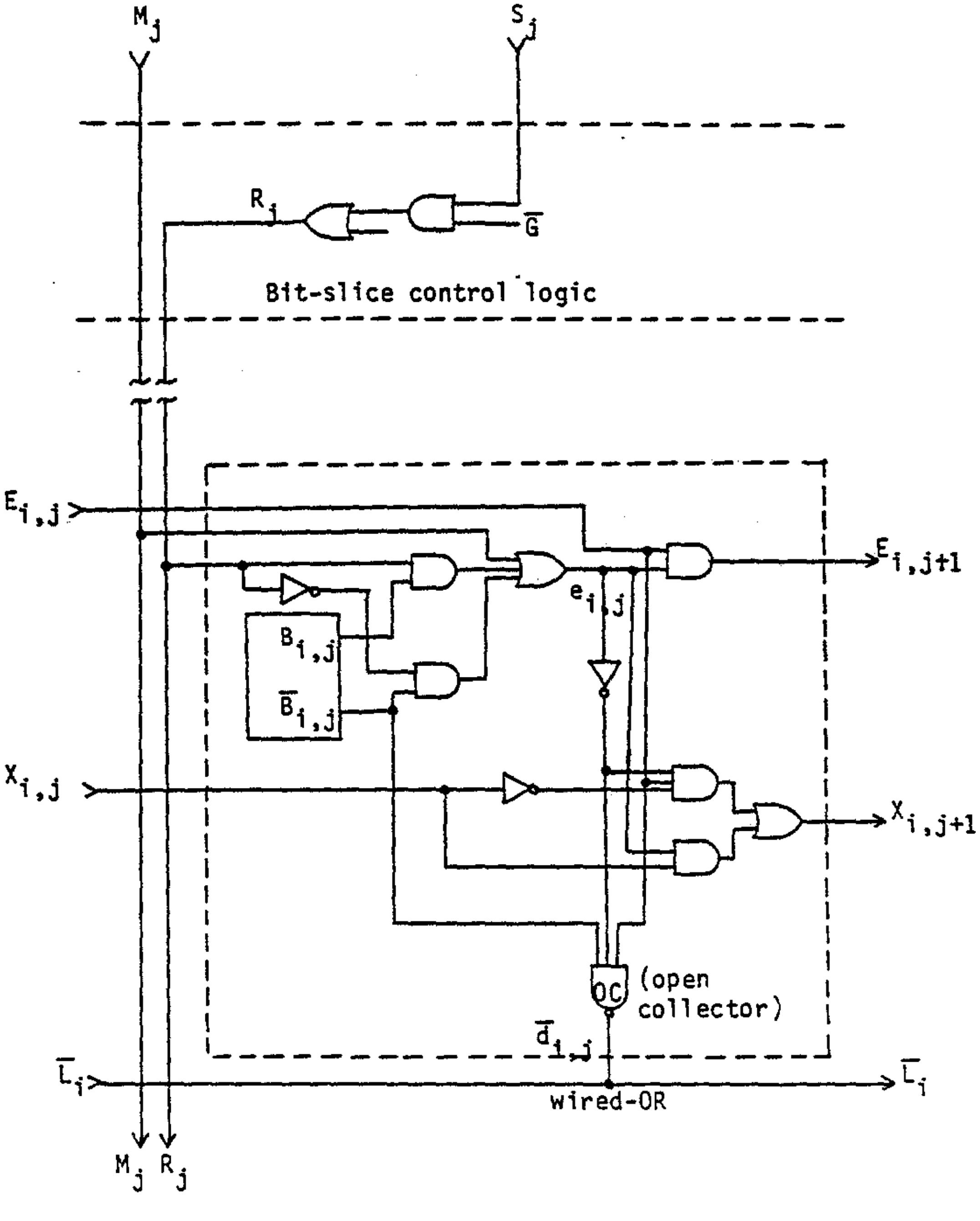


Figure 5.2 Bit-cell with Equality, Greater-than, Less-than and Proximity Capability for Mode A Operation

long.

Example 5.1: "Mode A" Search Operation

Search Word - S 10110,

Mask Word - M 00100,

Effective Search Word - S' 10d10 (d = don't care).

	i	B_i	State of $(L_i, E_{i,j+1}, X_{i,j+1})$ lines at the end of the minor cycle					
			j=0	1	2	3	4	5
memory words	1	10111	010	010	010	010	010	001
	2	11000	010	010	001	001	000	000
	3	10010	010	010	010	010	010	010
	4	10110	010	010	010	010	010	010
	5	10101	010	010	010	010	101	100
	6	01101	010	101	100	100	101	100

For interpretation of $L_i, E_{i,6}, X_{i,6}$ see step 5) of Algorithm 5.1 for Mode A search operations.

5.2.5.2 *Mode B: Least Value Search Mode*

In this mode, the search register is no longer needed because no search word is used. However, the minor cycle is more complicated than that in Mode A. It now consists of a comparison phase and a default phase. Consider the j'th minor cycle. In the comparison phase, one of tha three conditions is to be detected:

1) that the bit-slice is masked,

2) that the bit-slice is not masked and at least one enabled bit-cell contains a "0", and

3) that the bit-slice is not masked and all enabled bit-cells contain "1"s.

In the first case, all the enable signals to this bit-slice are passed on to the next bit-slice on the right. In the second case, those enabled bit-cells containing "0"s pass its enable signal to the next bit-cells on the right. In both cases, the minor cycle is complete. The third case, however, is called the default case and the default phase is entered. The default condition is detected in the default-detection bus and the default signal Q_j is fed back to the bit-slice via R_j. R_j is connected to the default feedback circuitry ($R_j = P_j \cdot \overline{Q_j} \cdot G$) when this search mode is activated by setting G = 1. P_j is a synchronization signal and it also serves as the search signal in the comparison phase. After the default phase, all the enabled bit-cells pass their enable signals to the next bit-cells on the right, thus completing the minor cycle. The result of Mode B can be stored in one of the result/temporary store registers because Mode A does not operate simultaneously with Mode B.

The implementation of the Mode B search in bit-cell (i,j) is shown in Fig. 5.3. Note that this implementation shares much of the circuitry with that for Mode A and that at the beginning of the j'th minor cycle, $R_j = 0$.

Algorithm 5.2 - Mode B Search Operation - The Least Value Search Algorithm

```
1) $\forall i \in \{1, 2, \ldots, m\}$
   a) Initialization:
      $G = 1$, $j = 0$, $w_{i,1} = 1$, $w_{i,2} = 0$,
   b) Data Path Setting:
      Gate $E_{i,n+1}$ to $w_{i,2}$, $w_{i,1}$ to $E_{i,1}$.
      The data paths and the control signal G are held until the completion of the major cycle.
2) Let $j \leftarrow j + 1$.
3) Minor Cycle:
   a) Comparison Phase: Compute $\forall i \in \{1, 2, \ldots, m\}$ simultaneously.
```

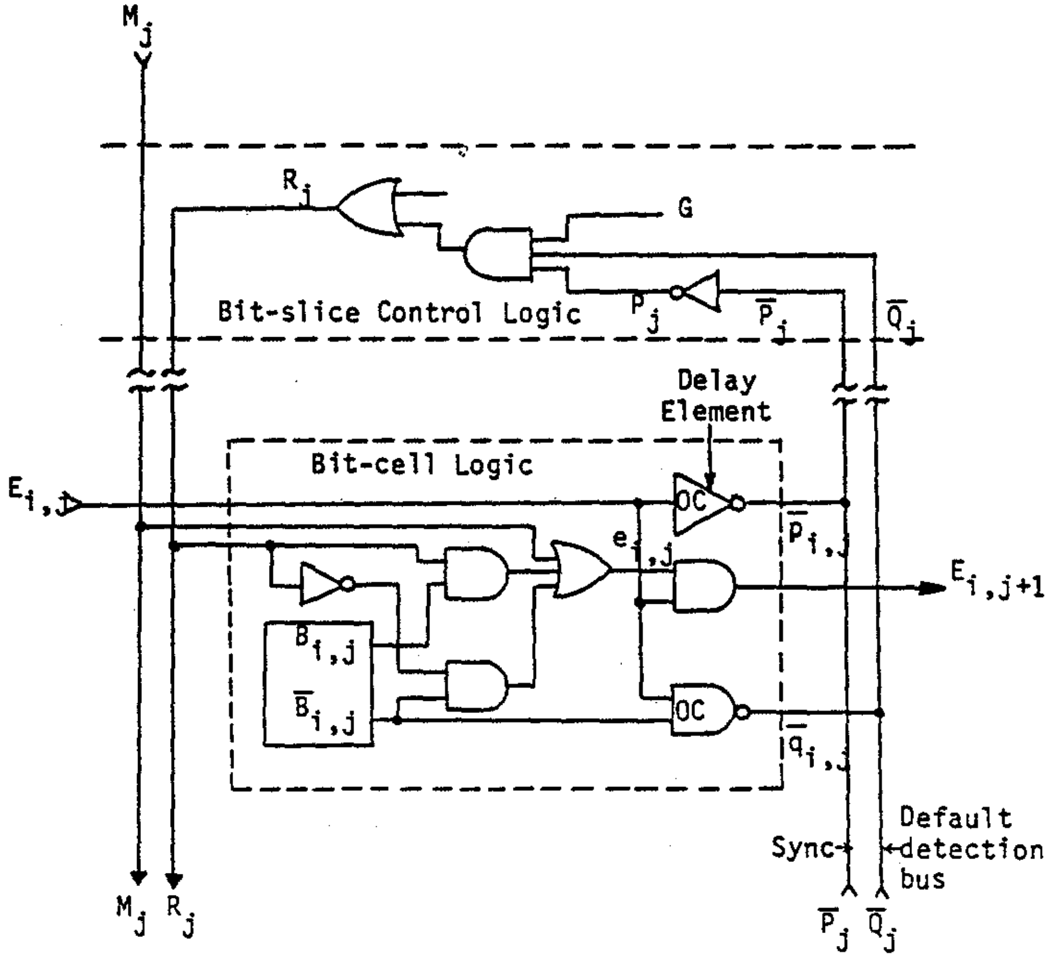


Figure 5.3 Bit-cell with Least Value Search Logic for Mode B Operation

i) $E_{i,j+1} = E_{i,j} \cdot e_{i,j}$.

ii) $p_{i,j}(t) = E_{i,j}(t - 2)$ (delay element used to synchronize the feedback of Q_j *via* R_j).

iii)

$q_{i,j} = E_{i,j} \cdot \overline{B}_{i,j}$,

$q_{i,j} = 1$ means that $B_{i,j}$ is enabled and equals 0.

iv)

$P_j = \bigcup_{i=1}^{m} p_{i,j}$ (wired-OR).

v) $Q_j = \bigcup_{i=1}^{m} q_{i,j}$ (wired-OR),

$Q_j = 1$ means at least one enabled bit in the j'th column is 0.

vi)

$R_j = P_j \cdot \overline{Q}_j \cdot G$.

b) Is $R_j = 1$?

i) Yes - Default detected, proceed to step 3c).

ii) No - Default inhibited, proceed to step 4).

c) *Default Phase:* Compute $E_{i,j+1} = E_{i,j}$.

4) Is j = n?

a) Yes - Proceed to step 5).

b) No - Proceed to step 2).

5) Read out the words that are indicated by $w_{i,2} = 1$.

Example 5.2 shows an example search of Mode B operation on 5 words, each 10 bits long.

Example 5.2 "Mode B" Search Operation

a) WORDS in which the least value is to be retrieved:

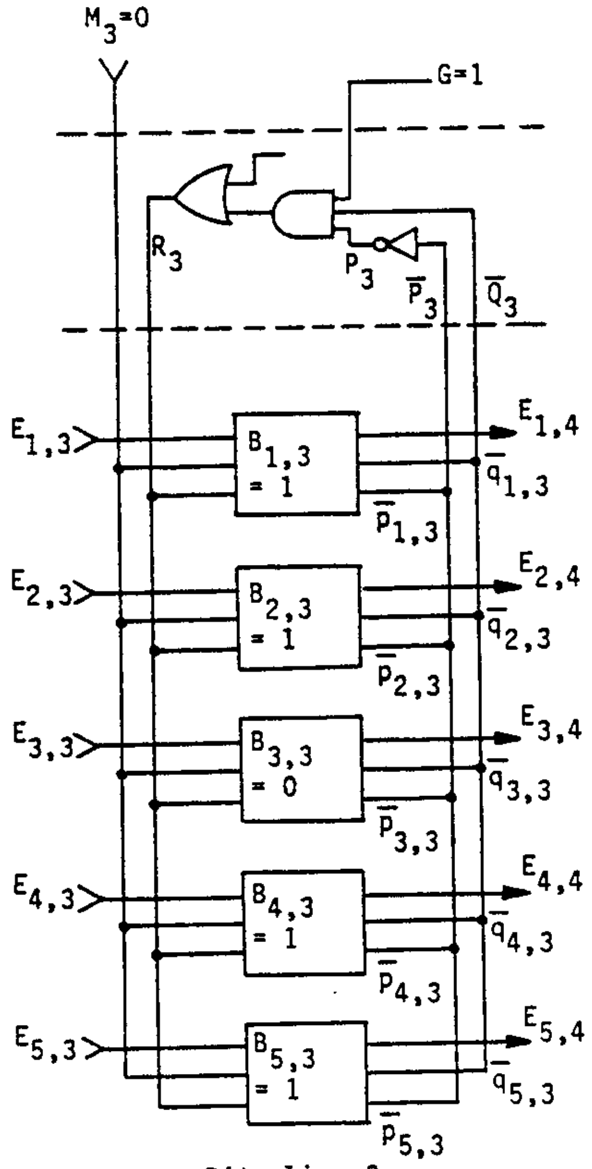
Word	Bit Positions										Order
Number	1	2	3	4	5	6	7	8	9	10	of
(i)						(j)					Retrieval
1	0	0	1	1	0	0	1	0	0	1	3
2	0	0	1	0	1	0	0	1	0	1	1
3	1	0	0	1	1	0	0	0	0	1	5
4	0	0	1	0	1	0	1	0	1	1	2
5	0	0	1	1	0	0	1	0	1	1	4

b) STATES of all enable lines ($E_{i,j+1}$) at the end of the major cycle:

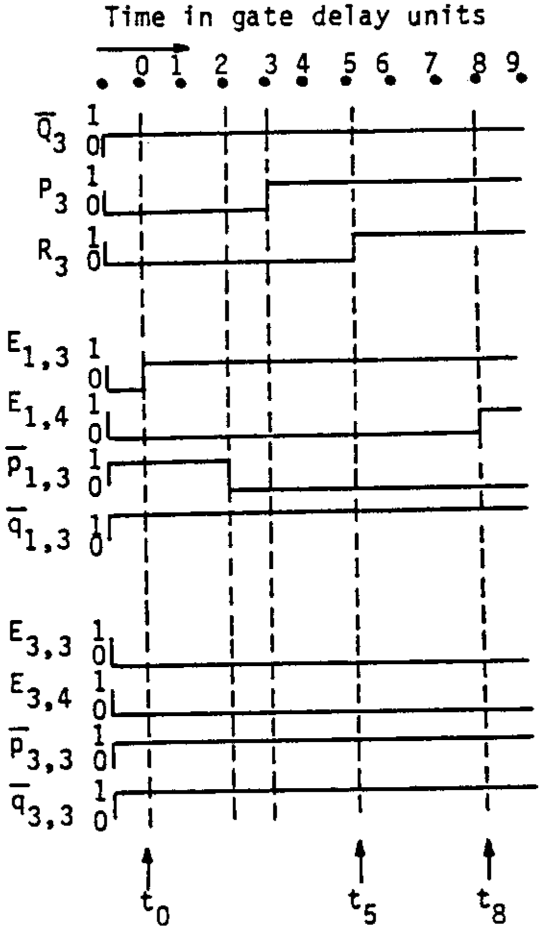
Word										
Number	0	1	2	3	4	5	6	7	8	9
(i)						(j)				
1	1	1	1	1	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1
3	1	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1	0	0	0
5	1	1	1	1	0	0	0	0	0	0

Note: Minor cycles 3, 5, 8 and 10 go through the default phase.

c) Timing diagram for bit-slice 3 in the major cycle:



Bit-slice 3



Note: Signals for bit-cells (2,3), (4,3) and (5,3) are the same as those for bit-cell (1,3).

t_0: Starting of minor cycle for bit-slice 3;

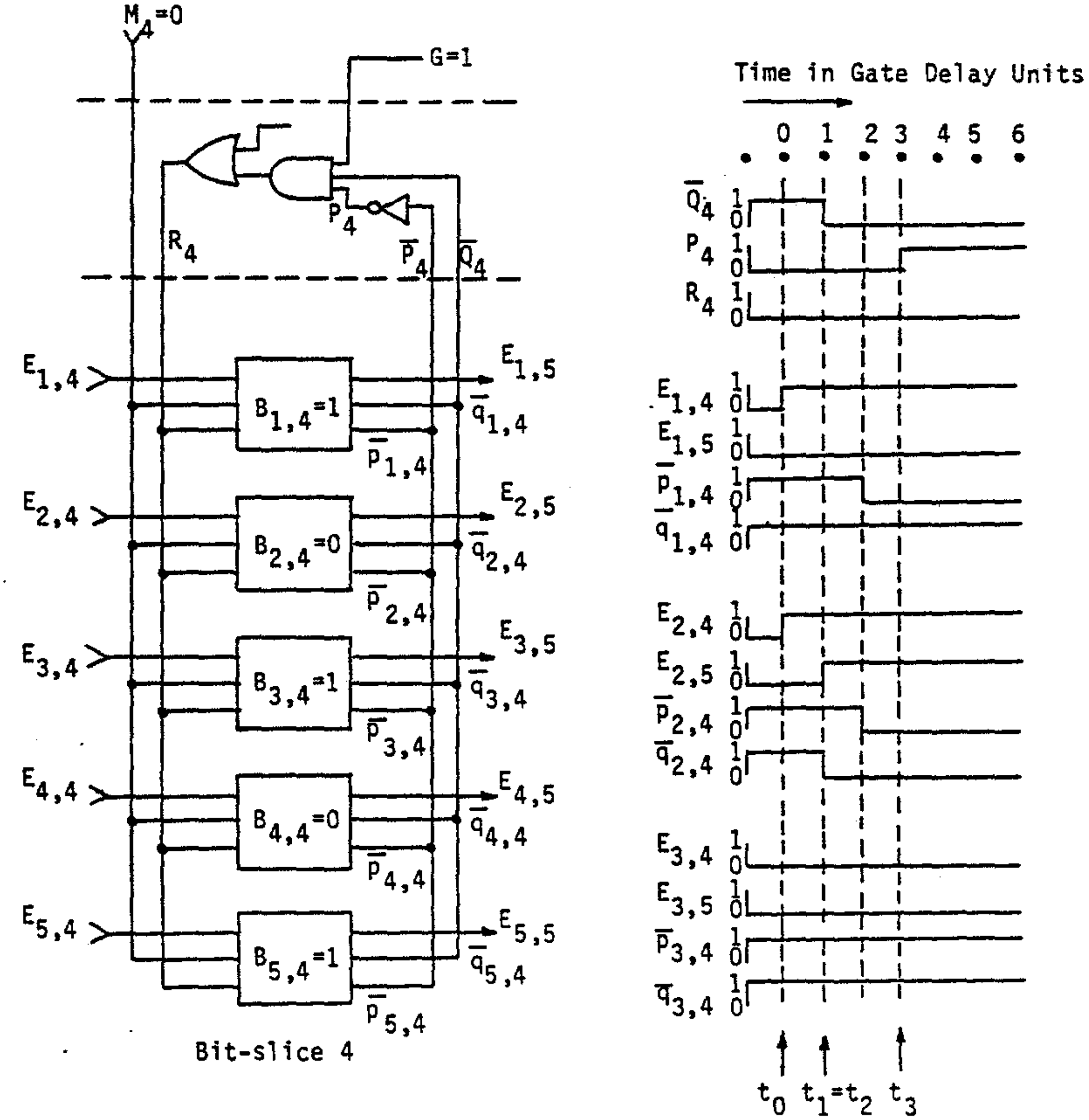
t_5: Default condition detected;

t_8: End of minor cycle for bit-slice 3;

FROM t_0 TO t_5: Comparison phase of minor cycle;

FROM t_5 TO t_8: Default phase of minor cycle.

(d) Timing Diagram for Bit-slice 4 in the Major Cycle



Note: Signals for bit-cells (4,4) and (5,4) are the same as those for bit-cells (2,4) and (1,4) respectively.

t_0: Starting of minor cycle for bit-slice 4;

t_1: End of minor cycle for bit-slice 4;

t_2: Default-inhibit signal becomes stable;

t_3: Bit-slice control logic in stable state.

5.2.5.3 *Mode C: Greatest Value Search Mode*

In the implementation of the least value search scheme, the speed for searching is traded for less hardware in each bit-cell by sharing much of the logic with the equality-inequality search. Had it not been required for the latter search, the comparison time for the least value search could be shortened by looking only at the content of the bit-cell, and the default time could be shortened by looking at the feedback signal. Since the least value and the greatest value searches are analogous to each other, we shall demonstrate the speed-up design for the greatest value search. The implementation of the new design is illustrated in Fig. 5.4 which shows the complete design for each bit-cell. With this implementation, Mode C operations can be executed simultaneously with either Mode A or Mode B operations. Note that $T_j = 0$ at the beginning of the j'th minor cycle.

Algorithm 5.3 - Mode C Search Operation - The Greatest Value Search Algorithm

1) $\forall i \in \{1, 2, \ldots, m\}$

   a) *Initialization:* j = 0, $w_{i,1} = 1$, $w_{i,5} = 0$,

   b) *Data Path Setting:* Gate $F_{i,n+1}$ to $w_{i,4}$, and $w_{i,1}$ to $F_{i,1}$,

   The data paths are held until the completion of the major cycle.

2) *Let* j ← j + 1.

3) *Minor Cycle:*

   a) *Comparison Phase:* Compute $\forall i \in \{1, 2, \ldots, m\}$ simultaneously.

      i) $F_{i,j+1} = F_{i,j} \cdot (M_j^{\bullet'} + B_{i,j} + T_j)$

      ii) $u_{i,j}(t) = F_{i,j}(t - 1)$ (delay element used to synchronize the feedback of V_j *via* T_j).

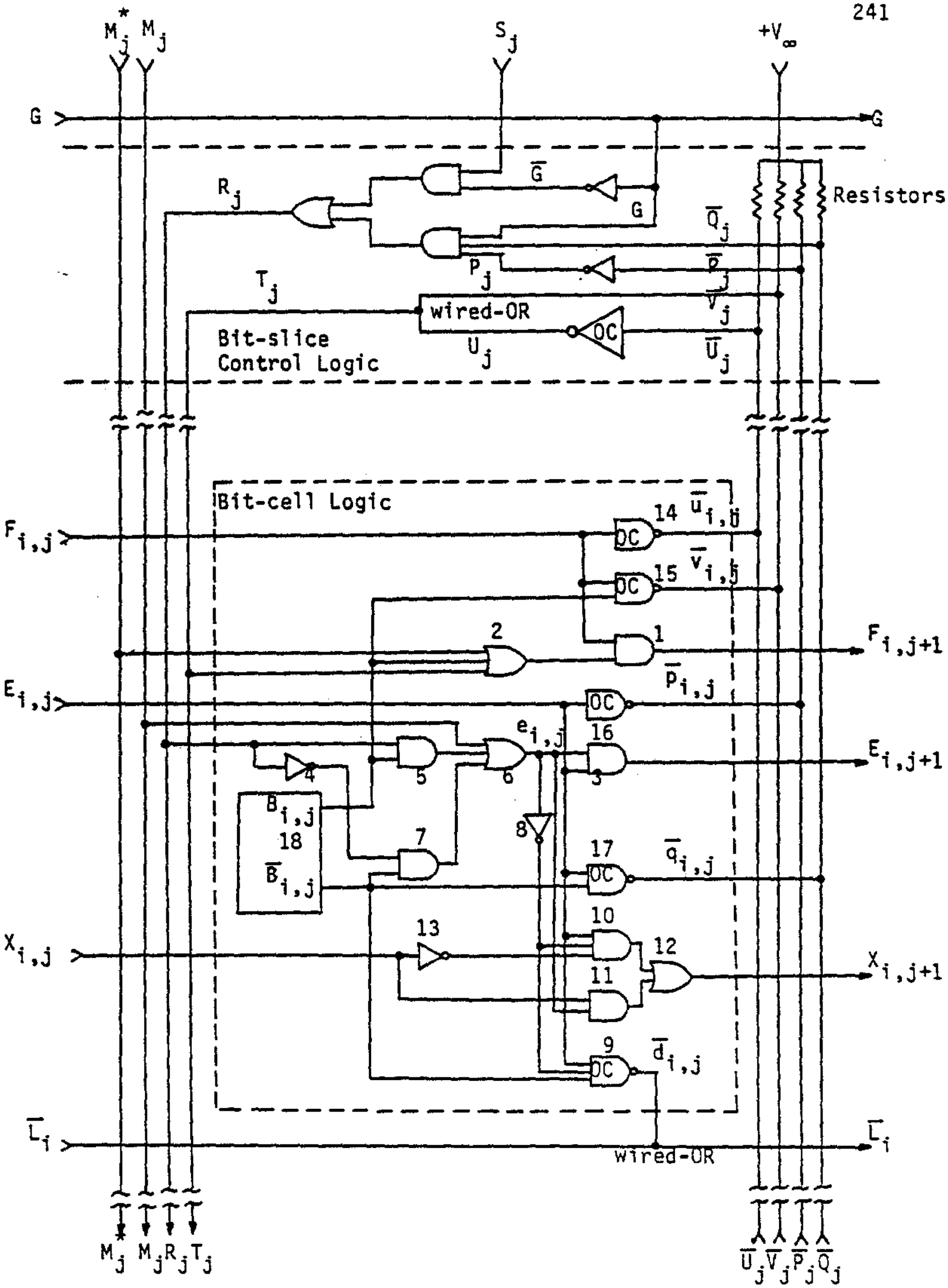


Figure 5.4 Bit-cell for Simultaneous Ascending Retrieval and Descending Retrieval or Equality-Threshold-Proximity Searches

iii)

$v_{i,j} = F_{i,j} \cdot B_{i,j}$,

$v_{i,j} = 1$ means that $B_{i,j}$ is enabled and equals 1.

iv)

$U_j = \bigcup_{i=1}^{m} u_{i,j}$ (wired-OR).

v) $V_j = \bigcup_{i=1}^{m} v_{i,j}$ (wired-OR).

$V_j = 1$ means at least one enabled bit in the j'th column is 1.

vi)

$T_j = U_j \overline{V_j}$ (wired-AND).

b) Is $T_j = 1$?

i) Yes - Default detected, proceed to step 3c).

ii) No - Default inhibited, proceed to step 4).

c) *Default Phase:* Compute $F_{i,j+1} = F_{i,j}$.

4) *Is* j = n?

a) Yes - Proceed to step 5).

b) No - Proceed to step 2).

5) Read out the words that are indicated by $w_{i,5} = 1$.

5.2.6. *Ordered Retrieval*

A. *Ascending Order Retrieval*

The ascending order retrieval of a set of data can be achieved by performing the least value search repeatedly until all the data are retrieved. With the CAM organization that we have presented, a microprogram in the Memory Control Store provides an economical and efficient implementation of such a retrieval algorithm. A flow chart for an ascending order retrieval algorithm is shown in Fig. 5.5.

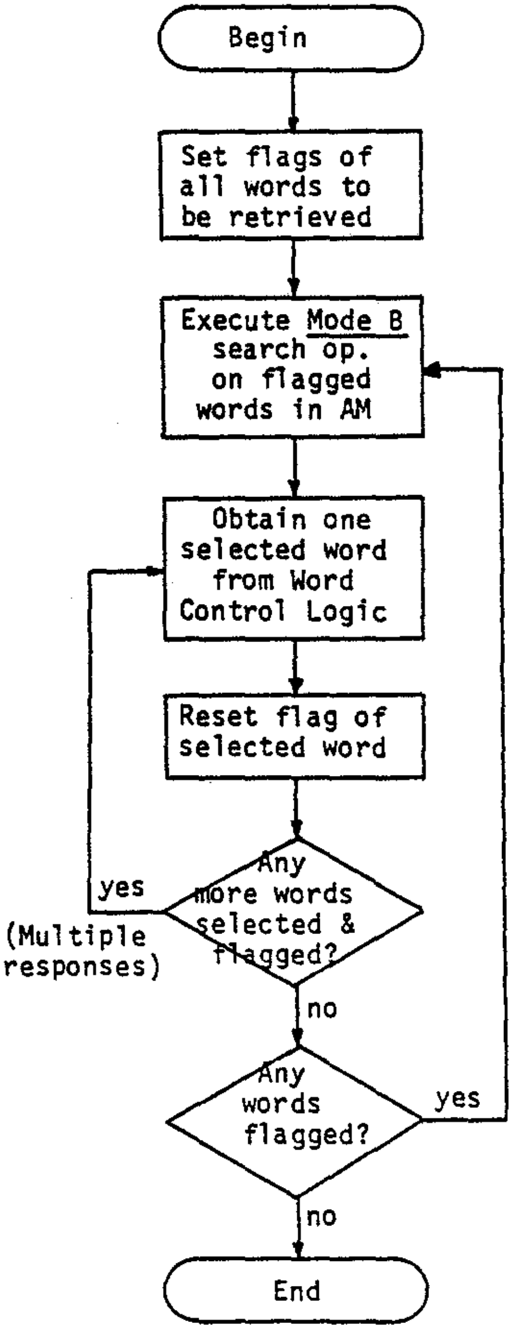

Begin
Set flags of all words to be retrieved
Execute Mode B search op. on flagged words in AM
Obtain one selected word from Word Control Logic
Reset flag of selected word
Any more words selected & flagged?
yes
(Multiple responses)
no
Any words flagged?
yes
no
End

Figure 5.5 Flow Chart for Ascending Order Retrieval

B. *Descending Order Retrieval*

The algorithm described for ascending order retrieval can be modified for descending order retrieval by substituting the greatest value search for the least value search. That is, Mode C search operation is executed in the CAM instead of Mode B search operation. Hence, the algorithm for descending order retrieval is to perform the greatest value search repeatedly until all the data are retrieved.

5.2.7 *Some Speed- up Techniques*

The design shown here have 1 to 3 gate delays per minor cycle in each of the Mode A search operations.[2] The delay in Mode B operations ranges from 3 to 8 gate delays per minor cycle,[3] while for Mode C operations, it ranges from 1 to 4 gate delays per minor cycle. We now consider several techniques that can be used to reduce the search times. The four areas that bear investigation are lookahead techniques, external examination of retrieval process, implementation of additional basic operations, and modifications to the scheme involving greater parallelism in the search.

In the first area, lookahead logic can be added to each word in the memory. The algorithms we have described previously are all bit-serial and word-parallel in nature. This means that the enable signals for each word propagate from bit to bit and operations for each word are performed in parallel. The speed of a search operation is therefore proportional to n where n is the number of bits in

[2] Assuming that all the $e_{i,j}$ signals are available before the search begins, there is one gate delay per minor cycle in the equality-threshold search while there are 2 to 3 gate delays per minor cycle for the propagation of the $X_{i,j}$ signal.

[3] In the case of the least value search, the maximum and the minimum delays are actually shorter. The $e_{i,j}$ signal of each bit-slice can be assumed to be settled before the major cycle starts (see Fig. 5.4). This means that the M_j lines are enabled ahead long enough for the $e_{i,j}$ signal to settle. In this case, the minimum time to pass through each bit-slice is 1 gate delay. The maximum time to pass through each bit-slice is also shorter than 8 (the maximum gate delay count). When default occurs, $B_{i,j}=1$ for all enabled words. Therefore, output from gate 7 is 0 and the feeddback through R_j never has to go through gate 4. Hence, the maximum delay through a bit-slice is 7 gate delays.

each word. We can increase the speed by adding some lookahead logic to each word. Each word is segmented into contiguous groups of bits of equal size k, and a lookahead circuit is added to each group (assuming k is a factor of n). Each lookahead circuit operates on all the bits in its group in parallel and passes the result onto the next group when it has finished. The speed of an equality-proximity search operation using this lookahead circuit will be proportional to $\frac{n}{k}$, but for extremum searches, no improvement is found. This type of lookahead is essentially single-leveled or cascaded. This means that the signals still have to propagate from group to group instead of from bit to bit, and the lookahead circuits exist in a single level above the storage circuits of each word. The cellular property of the design is preserved because a group, instead of a bit in a word, can now be regarded as a cell. We will not investigate other types of lookahead circuits, e.g., tree-lookahead circuits, because they do not preserve the cellular property. We now illustrate the construction of these lookahead circuits for the equality-proximity and the Mode B and Mode C searches.

An examination of the equality-proximity search operation shows that each of the $E_{i,j+1}$ signals propagates from bit slice j to bit slice j + 1 in one gate delay where j ranges from 1 to n. Similarly, the $X_{i,j}$ signal propagates from bit-slice j to bit-slice j+1 in 2 to 3 gate delays. Improvement can be achieved by grouping bits in each word and performing the comparisons in parallel. An example is shown in Fig. 5.6 where the necessary lookahead logic for grouping bits j and j + 1 of word i is shown. In the equality search, comparisons in each group are done in parallel. The results of comparison, $e_{i,j+1}$ and $e_{i,j+2}$, are ANDed together with $E_{i,j}$ to form $E_{i,j+2}$. The propagation time for these two bits is 1 gate delay instead of 2 in the usual bit serial operation. The speed of the equality search will therefore be proportional to $\frac{n}{2}$ gate delays. The number of gates for the propagation of $E_{i,j}$ signal is also reduced from 2 to 1 for bit-slices j and j+1.

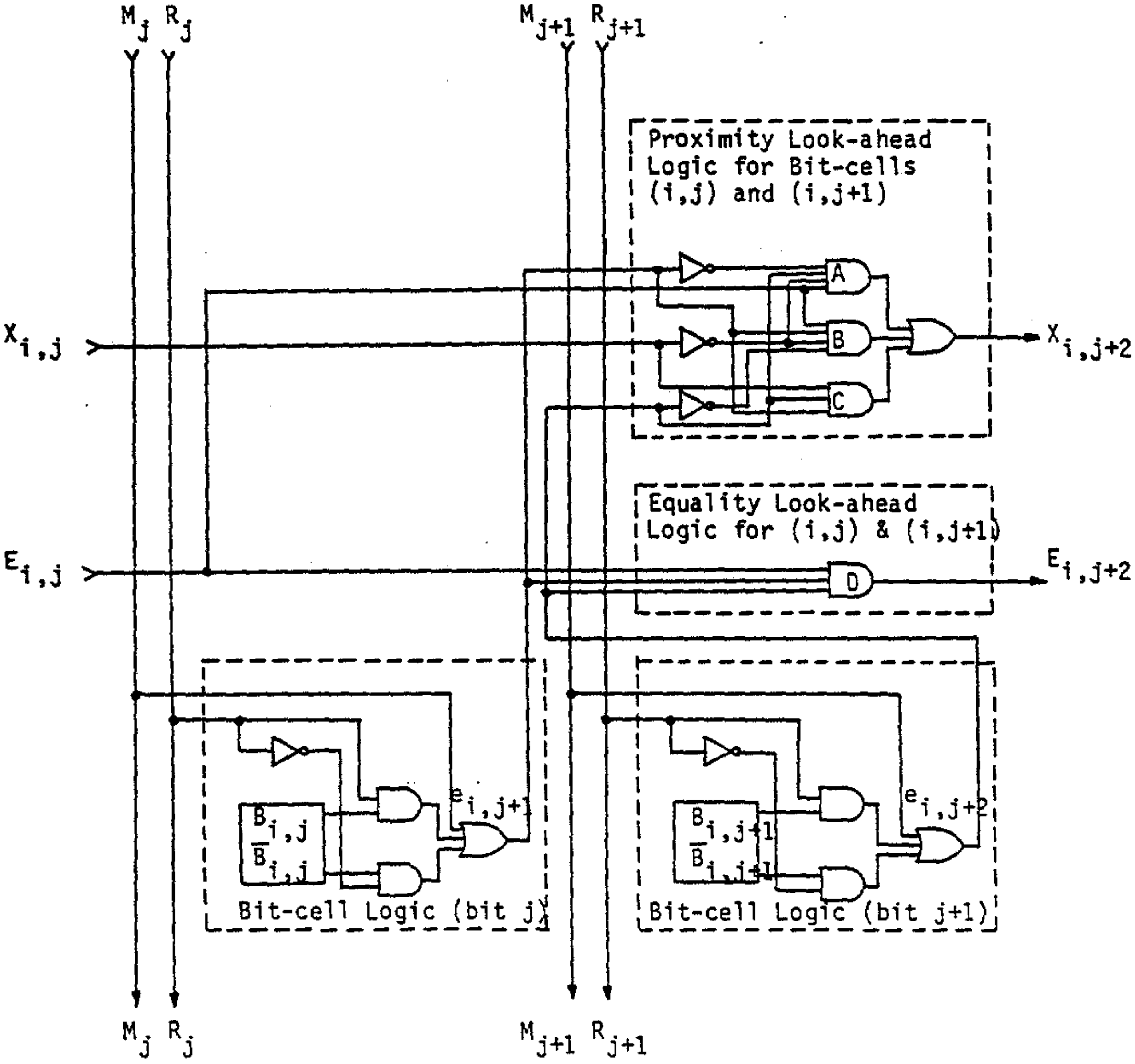


Figure 5.6 Bit-cells (i,j) and (i,j+1) of Word i with equality-proximity search logic and Look-ahead Logic

Similarly, in the proximity search, gates A and B of Figure 5.6 detect the condition when only one mismatch occurs in the group slice and gate C detects the condition when there is no mismatch in the group slice. The logic equation for $X_{i,j+2}$ is:

$$X_{i,j+2} = X_{i,j} \cdot (e_{i,j+1} \cdot e_{i,j+2}) + \overline{X}_{i,j} \cdot E_{i,j} \cdot (e_{i,j+1}, \overline{e}_{i,j+2} + \overline{e}_{i,j+1} \cdot e_{i,j+2})$$

which is similar to the $X_{i,j+1}$ equation in Algorithm 4.1. However, in this case, the propagation delay has been reduced to 3 instead of 6. The number of gates required is also reduced by a constant factor.

For Mode B and Mode C operations, lookahead requires more hardware. The existence of default cases have caused the increased complexity. Previously, without lookahead, default is detected for a bit-slice when certain conditions exist on all enabled words in that bit-slice. These conditions include 1) all enabled words have 1's in this bit-slice for the least value search and 2) all enabled words have 0's in this bit-slice for the greatest value search. The number of default feedback lines is 1 for each search mode. When a lookahead circuit is added to each word for a group of k bit-slices, the number of default feedback lines will be 2^k. These 2^k lines can be shared by both the least value search and the greatest value search. Consider a particular group; the following operations are to be carried out: a) The bits of each word in this group are decoded into 2^k lines. b) The corresponding lines from each word of this group are wired-ORed together to form default feedback lines 0 to $2^k - 1$; a particular feedback line p will be 1 when there exists an enabled word in this group whose decoded value equals p. c) In the group-slice control logic, if it is a Mode B operation, it will scan from feedback lines 0 to $2^k - 1$ until the first line with a 1 is found; similarly if it is a Mode C operation, it will scan from feedback lines $2^k - 1$ to 0; this line will represent the minimum/maximum of all these enabled words in this group. d) This line is encoded into k search bus signals to be fed back to each word in this group. e) In a particular word, the enabled line for the next group is enabled if the current group of this word is enabled and the value of this part of the word equals the search bus signal, i.e., it equals the minimum/maximum value found by the group slice control logic. However, there are some disadvantages of using lookahead on Mode B and Mode C operations. The extensive amount of decoding requires an order of 2^k gates of fan-in k for each group in each word. For each group-slice, there are 2^k default feedback buses running across all the words and this can cause difficulty in integrated circuit implementation. The biggest difficulty, however, lies in the implementation of the scanning algorithm in the group-slice control logic. The algorithm of scanning across a set of lines until the first 1 is found is essentially a multiple match resolution problem. If a tree-type multiple match resolution circuit is used, e.g., [FOS68], a maximum delay of $\log_2 2^k = k$ will be observed. That is, the overall speed of a group of bit slices, with or without lookahead, is of the order of k. Unless a faster multiple match resolution circuit is used, and the cost of hardware is sufficiently low, lookahead for Mode B and Mode C searches is not cost-effective.

An examination of the example illustrated in the previous section points out another possible source of improvement, this time in the algorithm itself. In many cases the number of words still enabled at the end of a minor cycle rapidly drops to one within a few minor cycles. At this point the completion of the major cycle is a formality since the greatest(or the least) valued word must be the only remaining enabled word. Unfortunately the detection of this condition, the only-one-respondant-left condition, is too complex to be performed at the end of every minor cycle, and would require extensive external wiring and logic.

We have implemented some of the search operations defined in Section 5.2.4 as basic operations. Some other useful searches may be performed by combining two or more basic searches and possibly some nonsearch operations.

An example is the between-the-limits searches, which is generated by performing a less-than search followed by a greater-than search on words selected by the first search. In fact, all the searches described in Section 5.2.4, can be performed as a basic search or a combination of basic searches designed in this section. Speed improvements can of course be gained by implementing all of these search operations as basic searches, but the amount of logic circuits may be extensive. In most other cases, the more complicated searches, such as the case of ordered retrieval, are implemented as a combination of simple searches.

One modification to our ordered retrieval technique that yields positive results without compromising our cellular logic approach is to increase the parallelism of the algorithm itself. This can be done by simultaneously performing the greatest value search and the least value search on the same set of enabled words. The associative sort is complete when both searches select the same word, an easily detectable condition, or when no words are still enabled at the beginning of a major cycle, also an easily detectable condition. A small additional amount of external manipulation of the sorted file block is required by the non-associative processor controlling the sort to concatenate the two halves of the sorted block since one will be in the reverse of the desired order, but it is felt that this is a small price to pay for a speed-up factor of greater than 2. This technique is shown in Example 5.3 that follows.

The speed-up involved in this approach is greater than a factor of 2. To understand why it is greater than a factor of 2 instead of exactly equal to 2, we must consider the properties of the fields to be searched. Assuming an even distribution, there is on the average one more bit with the value "1" in the higher valued half of a sorted file than in the lower valued half of the same file. This can be verified in Example 5.3a). The greatest value search has a shorter minor cycle time for bit positions with a value of "1" in the word with the greatest value than for bit positions with a "0" in the word with the greatest value. Likewise, the least value search has a shorter minor cycle time for bit positions with a value of "0" in the word with the least value than for bit positions with a "1" in the word with the least value. This provides for an average major cycle time five gate delays shorter than if all words were to be selected in an ordered retrieval by either search alone (assuming the delay for each minor cycle of both Mode B and Mode C search operations ranges from 3 to 7 gate delays). The design for this technique has been indicated in Fig. 5.4.

Example 5.3: "Mode B" and "Mode C" Parallel Operation.

a) WORDS to be retrieved:

Word											A = Number	Ascending
	Bit Positions										of 1's	Order of
Number	1	2	3	4	5	6	7	8	9	10	per Word	Retrieval
1	0	0	1	0	1	0	1	1	0	1	5	12
2	1	1	0	0	1	0	0	1	0	0	4	20
3	0	0	0	0	1	1	1	1	0	1	5	6
4	1	1	1	1	1	1	0	0	0	0	6	30
5	0	0	1	0	0	0	0	0	0	0	1	11
6	1	1	1	0	0	0	0	1	0	0	4	23
7	0	0	0	0	0	0	0	0	0	0	0	1
8	0	1	0	1	1	1	1	1	0	1	7	14
9	1	1	0	1	0	1	0	0	1	0	5	21
10	1	1	1	1	0	0	1	1	0	0	6	28
11	1	1	0	1	1	1	1	1	1	1	9	22
12	0	0	0	1	1	1	1	0	1	1	6	10
13	0	0	0	0	0	1	1	1	1	1	5	4
14	0	1	1	0	1	0	1	1	1	1	7	15
15	1	1	1	1	1	1	0	1	1	0	8	31
16	1	1	1	0	1	0	0	0	0	0	4	26
17	1	1	1	1	1	0	0	0	0	0	5	29
18	1	1	1	0	0	0	1	0	0	1	5	24
19	0	0	0	0	0	0	1	0	0	1	2	2
20	1	0	1	0	0	0	0	0	1	0	3	19
21	0	0	1	1	0	1	1	0	1	1	6	13
22	0	0	0	1	0	1	1	1	1	1	6	7
23	1	1	1	1	1	1	1	1	1	1	10	32
24	0	0	0	0	0	0	1	1	1	1	4	33
25	0	1	1	1	1	1	0	1	1	0	7	16
26	0	0	0	1	1	0	1	0	1	1	5	8
27	1	1	1	0	0	1	0	1	0	0	5	25
28	0	0	0	1	1	1	0	1	1	0	5	9
29	1	1	1	1	0	0	0	0	1	0	5	27
30	1	0	0	1	0	1	0	0	0	0	3	18
31	0	0	0	0	1	1	0	0	1	1	4	5
32	1	0	0	0	0	0	1	0	0	1	3	17

Number of 1's in memory: 160

Number of bits in memory: 320

Number of 1's per word in the smaller half of the ordered list = 4.69.

Number of 1's per word in the larger half of the ordered list = 5.31.

b) ORDER of retrieval in parallel operation:

Let L_B and L_C be the lists of words retrieved by Mode B and Mode C search operations, respectively. Both lists are ordered with respect to time, in Gate Delay Units, at which they are retrieved, and neglecting overhead time between major cycles. Assume that for the least value search, the gate delays for each minor cycle range from 1 to 7 and that for the greatest value search, they range from 1 to 4.

L_B	*Time*[a]	L_C
Start	0	Start
7	10	23
-	26	15
19	32	-
-	48	4
24	66	-
-	73	17
-	95	10
13	106	-
-	120	29
31	140	-
-	148	16
-	173	27
3	180	-
-	198	18
22	226	6
-	239	11
-	264	9
26	266	-
-	292	2
28	306	-
-	323	20
12	352	-
-	354	30
5	368	-
-	385	32
-	404	25
1	408	-
-	423	14
-	442	8
21	454	-
End of Retrieval		

[a] Time in Gate Delay Units

Throughput = 454 gate delays (32 * 10) bits = 1.42 gate delays/bit.

5.2.8 *Issues and Limitations*

We have presented a design of an associative memory that can be used for fast ordered retrieval. From Example 5.3, neglecting the overhead in loading and unloading the memory, the sorting speed is 1.42 gate delays per bit. This design is therefore very attractive and can be used in many places where fast searching and sorting is required. However, there exists many issues that need to be carefully considered and resolved before successful operations can result. We discuss four of these issues here, namely, LSI implementation, manufacturing defects, modular expansion, and multiple match resolution. We do not contend that they exhaust all the issues in this design. New issues may come up during the implementation phase and will have to be resolved by the designer.

5.2.8.1. *LSI Implementation*

In Fig. 5.4, a complete design has been shown. Each bit cell requires 17 gates. There are extra logic associated with the registers and the controls.

Consider a 32-bit word and a 32-word memory. This design needs over 17,000 gates for the logic in the bit-cells only, excluding all other registers, memory cells and control logic. Therefore, the memory size that can be effectively implemented on an LSI chip is very limited. One solution is to reduce the number of functions in a cell when the application does not call for it. However, this is very much application dependent. Furthermore, the number of pins on the LSI package also limits the word size. In order to maintain fast response and high throughput, parallel reading and writing of bits of a word in the memory is necessary. The major portion of the pins of an LSI package is usually taken up for parallel reading and writing. For a 32-bit word memory, the pin requirement is 32 plus a few controls and selections. On the other hand, the pin limitation will put a maximum word size that can be implemented. It becomes obvious that modular expansion is necessary in order for this design to be practical. The issue of modular expansion is discussed later.

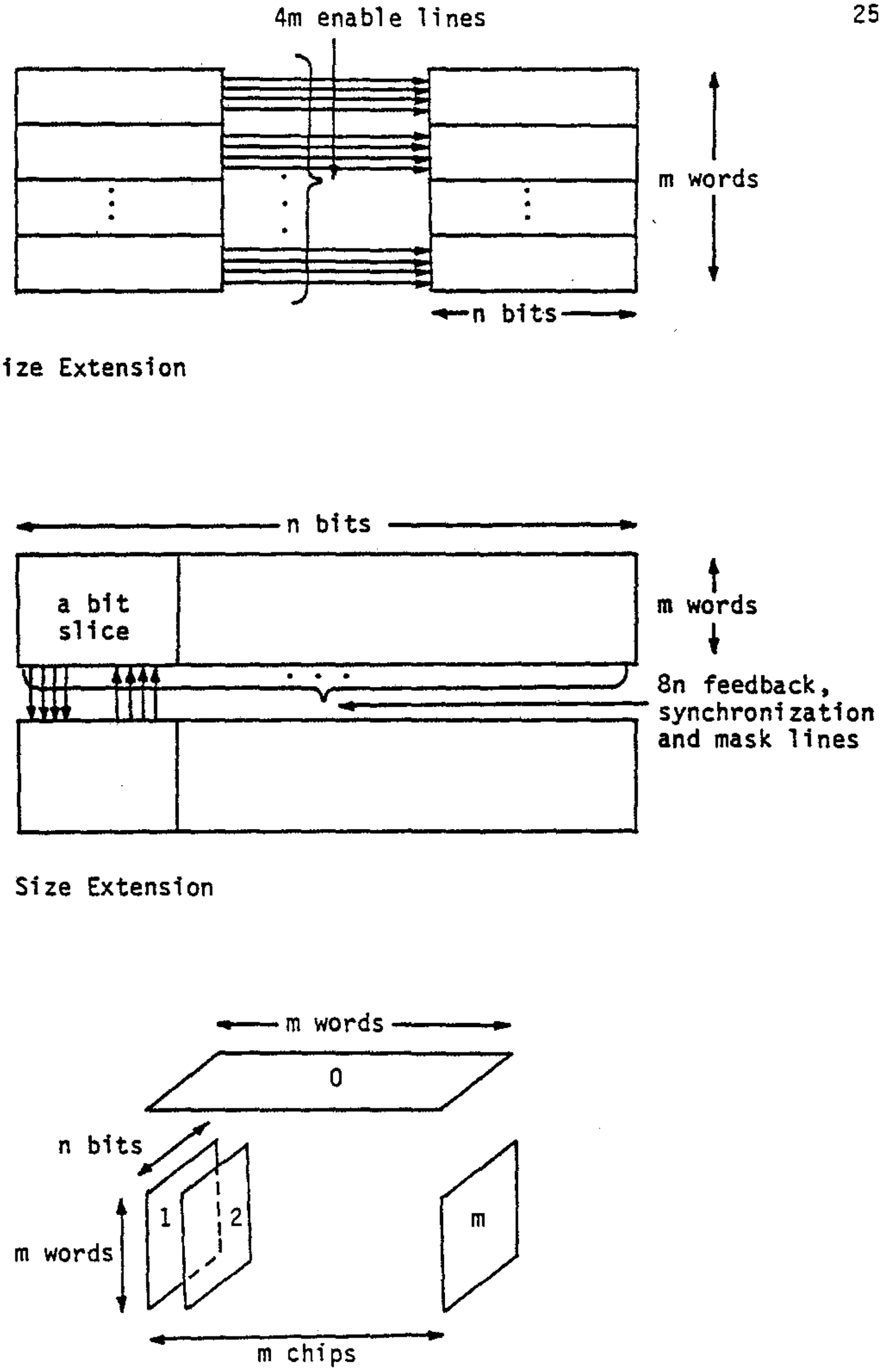
5.2.8.2. *Manufacturing Defects*

After the LSI chip has been manufactured, tests are made to determine whether any cells are faulty. A faulty cell can be determined by injecting certain test patterns into the memory. If the number of defects are small and their locations can be determined up to the locality of certain gates in the cell, then these faults can be bypassed by utilizing some spare bit-slices designed into the memory. The difficulty in recovering an error in a faulty cell of the CAM is that the error may not only affect the word itself, but it may also affect other words because the value of the faulty bit is available to other words via the feedback circuitry. Therefore, it may be necessary to remove the current bit-slice or the current and all bit-slices to the right from operation when an error occurs in a cell. We have assumed that only stuck-at faults can occur in the gates of memory cells and bit-slice control logic. Faults occurring in registers and control store are not considered since the logic there is only a small fraction of all the logic on the chip. By assuming that the j'th bit of the i'th word is faulty, we can identify three types of faults, one in which the j'th bit-slice has to be removed from operation, one in which the i'th word has to be removed from operation and one in which all the remaining bit-slices are rendered useless. Referring to Fig. 5.4, for faults that occur in gates 14-17 and the bit-slice control logic, they only affect the feedback values but they do not affect the enable lines so long as the mask bit is 1, that is, the bit-slice is masked off. This can be done by setting a 1 permanently in the j'th bit of the mask registers and shifting the external pin connection to the chip by 1 bit. For faults (stuck at 0 or stuck at 1) that occur at gates 2, 4-13 and the storage cell 18, and for stuck at 0 faults at gates 1 and 3, they do not affect the remaining words so long as the enable signals are set to 0, that is, the i'th word is disabled. This can be done by setting a 0 permanently in the i'th position of the word flag register W_1 and the result/temporary registers $W_2 - W_5$. For stuck at 1 faults that occur at gates 1 and 3, they affect the enable lines for the next bit-slice. If an enable line has a faulty value of 1, that is, the remaining bits of this word are enabled regardless of whether the current word or bit-slice are masked off, it may cause a faulty feedback to other bit-slices on the right. So unless all the remaining bit-slice are masked off, the fault that occurs in cell (i,j) will propagate to these bit-slices. A finer recovery procedure can be developed if we can identify the corresponding words to be disabled for a particular search operation.

From the above discussion, we see that recovery from manufacturing defects are easy and most of the faults are recoverable.

5.2.8.3. *Modular Expansion*

Our philosophy of the associative memory design is that we want to distribute the logic into the storage cells. In order for all the distributed logic to perform coherently, extra communication lines are needed to transfer enable and feedback signals from bit to bit. The number of these communication lines are usually large and this will eliminate the possibility of modular expansion which is easy in the case of RAM's. Consider our design in Fig. 5.4, each cell has 4 enable lines to communicate with the cell on its right; and each bit-slice has 8 lines which are used for feedback, synchronization and mask. These lines run across all words in the bit-slice (these exclude lines needed to read and write data into each bit). Suppose a memory chip of m words by n bits is available. To extend the word size of this memory, we can put 2 memory chips together side by side as shown in Fig. 5.7(a). However, this design needs 4m lines to pass the enable signals from the chip on the left to the one on the right. This is not feasible even for a small m. To extend the memory size, we can put 2 chips one over the other as shown in Fig. 5.7(b). This design needs 8n feedback lines to pass the feedback, synchronization and mask signals between the two chips. Even for a small value of n, the number of interconnections is very large. In order for our design to be practical, some other schemes of modular expansion are necessary. In Fig. 5.7(c), we show a scheme that allows us to extend the memory size by increasing the dimensions of the memory. A batch of m memory chips are put together in parallel. There is an extra dimension and is composed of a single memory chip running across the m parallel chips. A flow chart for an ascending order retrieval algorithm of m^2 words is shown in Fig. 5.8. The time needed to orderly retrieve m^2 words is $m^2 + m$ units of load time (time to store a word into the memory) and $m^2 + m$ units of search time (a search time includes the time to execute a Mode C operation and to read it out into the I/O register). The amount of search time can be reduced to m^2+1 units of search time when the Mode C searches in chips 1, ..., m are performed in parallel with the Mode C searches in chip 0. In a single memory chip which can accomodate m^2 words, the time needed for this memory system is m^2 units of load and m^2 units of



(a) Word Size Extension

(b) Memory Size Extension

(c) 3-dimensional Associative Memory for Memory Size Extension

Figure 5.7 Modular Extension for Proposed Associative Memory

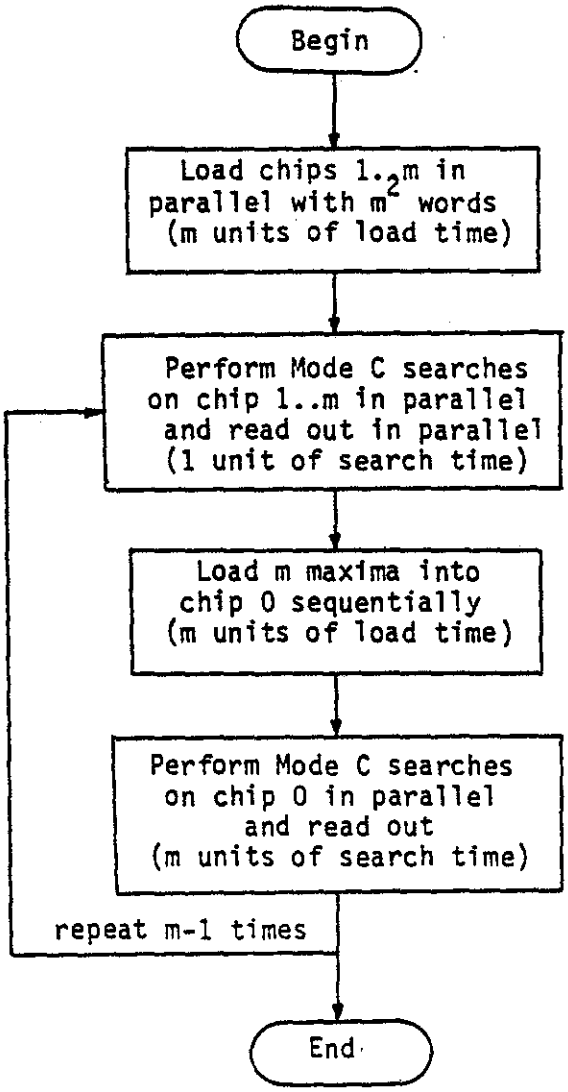

Begin
Load chips 1..m in parallel with m2 words (m units of load time)
Perform Mode C searches on chip 1..m in parallel and read out in parallel (1 unit of search time)
Load m maxima into chip 0 sequentially (m units of load time)
Perform Mode C searches on chip 0 in parallel and read out (m units of search time)
repeat m-1 times
End

Figure 5.8 Flow Chart for Ascending Order Retrieval of m^2 words in a Three-dimensional Associative Memory (see Figure 5.7c)

search time. Therefore the degradation in performance is minimal when m is large. For a memory size larger than m^2 words, extra dimensions are needed.

We conclude that our scheme on memory expansion has minimal degradation on performance. The difficulty still exists in word size expansion. The limitation is due to the pin requirements. However, we can trade performance for a smaller amount of external pin connections by loading bits of a word in groups instead of all in parallel. However, the degradation in performance due to this loading scheme is more pronounced than our memory size expansion scheme.

5.2.8.4. *Multiple Match Resolution*

One of the most useful applications in our design is in the multiple response resolution. A tag field can be included in each word. Each tag is a distinguishable number. The size of each tag must be at least $\lceil \log_2 m \rceil$ for a memory size m. When there are multiple responses, each tag serves as a number for the ordered retrieval scheme. The words used in the ordered retrieval are those that respond. Only the bit-slices containing the tag are used in the search. The first cycle can retrieve 2 words, the one with the maximum tag, and the one with the minimum tag. Subsequent searches give 2 responses each time. The speed of this resolution scheme is $\frac{1}{2}$ memory cycle per word and is independent of the memory size.

There are two disadvantages in using tags for multiple match resolution. First, there are irregularities in implementation. Because each tag has a distinguishable value and if each tag is hardwired into the memory, it will involve a different design for each word and it will also be difficult to overcome the problem of manufacturing defects when a cell in the tag is bad. This problem can be solved by loading the tags from a PROM when the memory is first used. Second, when a cell in the tag becomes bad during operation, e.g., stuck at 0, then two of the words in the memory have identical tags and it is impossible to distinguish them.

We can also perform the multiple match resolution without using special fields as tags. This can be done by treating the contents of each word or part of the word as a tag itself. It requires all words under consideration in the memory to be different in order for unique responses to result.

5.2.9 *Comparisons with Other Methods of Ordered Retrieval*

We have presented in this section several of the search schemes, namely, the equality search, the threshold searches, the proximity search, and the extremum searches. The other searches defined in Section 5.2.4, can be implemented as a combination of basic searches. Using the implementation in this section, we compute the maximum and the minimum search times for each search.

Search Type	Minimum Number of Gate Delays	Maximum Number of Gate Delays
Equality Search	n + 5	n + 5
Inequality Search	7	n + 5
Similarity Search	n + 5	n + 5
Greater-than Search	n + 5	n + 5
Less-than Search	7	n + 5
Greater-than-or-equal-to Search	n + 5	n + 5
Less-than-or-equal-to Search	n + 5	n + 5
Double-limit Search		
Between-limit Search, X > Y		
< X & > Y	n + 12	2n + 10
< X & ≧ Y	n + 12	2n + 10
≦ X & > Y	2n + 10	2n + 10
≦*X* & ≧ Y	2n + 10	2n + 10
Outside-limit Search, X < Y		
< X & > Y	7	2n + 10
< X & ≧ Y	7	2n + 10
≦*X* & > Y	n + 5	2n + 10
≦*X* & ≧ Y	n + 5	2n + 10
Proximity Search	2n+4	3n+6
Extremum Search		
1) Least-Value Search	n	7n
2) Greatest-Value Search	n	4n
Adjacency Search		
1) Nearest-above Search	2n + 5	8n + 5
2) Nearest-below Search	n + 7	5n + 5

We see that the delay times in all these searches are proportional to n, the number of bits in a word and is independent of the number of words in the memory.

Several methods of ordered retrieval and multiple response resolution have been proposed in the past. It would be of great value to evaluate the method of ordered retrieval presented in this section in terms of these other schemes. In particular, we compare this new algorithm with those of Frei and Goldberg [FRE61], Seeber and Lindquist [SEE62], Lewin [LEW62], Miiller [MII64], and Foster [FOS76]. In order to evaluate these various schemes, it is necessary to determine the significant characteristics that we wish to examine and to determine the comparable features of these diverse methods.

In order to facilitate these comparisons, the methods mentioned will be classified into two types, those with an algorithm to order the retrieval according to the contents of the stored words and those which use an external priority scheme, usually some form of priority tree, to order the retrieval according to the physical location in memory. Among schemes of the first type are those of Frei and Goldberg, Seeber and Lindquist, Miiller and Lewin. Miiller's scheme uses the contents of the responding words to resolve multiple response conflicts but it does not necessarily order the selections in ascending or descending order. Among those schemes that use an external priority circuit to resolve conflicts are those of Weinstein [WEI63] and Foster. These schemes are not strictly comparable to the proposed algorithm since they cannot be used for sorting. Likewise, Miiller scheme is not absolutely comparable to our proposed scheme but is similar enough that we will include it in the comparison.

The two main considerations for comparison are obviously the speed with which a method retrieves stored data and the cost in terms of amount of logic required. Rather than attempting an exhaustive analysis of the implementation cost for each of the various schemes, we shall look at the more readily available information as to the rate of cost increase for increasing memory size. In particular we are interested in the memory cost as a function of memory size.

We shall limit our discussion of speed comparisons to the number of search cycles required to retrieve each stored word. For several of the schemes under consideration, a significant parameter is the density of the flagged words, that is, the ratio of the number of words to be retrieved to the number of words addressable with the given tag field size. We will assume that the number of words addressable by the tag field is the same as the length of the memory.

The chart of Table 5.1 shows as direct a comparison as possible between the aforementioned searches and the search scheme proposed. The headings include relative speed (in terms of the number of cycles needed to retrieve each flagged word), comments upon dependencies of logic complexities to memory

Scheme	Speed - R (n,k) (Cycles per Retrieval of an n Bit Tag)	Memory Size Dependency	Relative Complexity of Hardware Required	Best Class of Problem
Frei and Goldberg	For n = 5 Best Case (k = 2^5): R = 2k Worst Case (k=1): R = $7k$	Length of tag field (n) only	Basic CAM	High density of responders
Seeber and Lindquist	$t = 2^n$ $R(n,k) = \frac{1}{k}\{(t+k-1) +$ $\frac{t(t-1)^k}{t^k} - \frac{2t}{t^k}\sum_{i=1}^{n}(k(t-2^i)^{k-1} +$ $2^{-i}(t-2^i)^k)\}$	Length of tag field (n) only	Complex cryogenic logic at each bit (18 gates)	Density dependent
Lewin	$\frac{2k-1}{k}$ *Exact*	Independent	A registers plus 9 gates per bit slice	Independent
Miller	*Best case* : $\frac{k+1}{k}$ *Worst case* : $\frac{2k-1}{k}$	Independent	Basic CAM plus some additional control and storage	Multiple response resolution only
Foster	1 cycle per retrieval	External logic uses $3(2^{m-1})-1$ gates for 2^m words of memory	Tree Circuit external to CAM	Multiple response resolution only
Proposed	$\frac{1}{2}$ cycle per retrieval	Increases as $\log_2 m$ * for m words of memory	~ 17 gates per bit cell	Independent

* $\log_2 m$ is the size of a tag that must be used to uniquely identify each word for a memory size of m. This differs from the other schemes which do not use a specialized tag for ordered retrieval.

Table 5.1 A Comparison Table for Ordered Retrieval Schemes

size, relative complexities of the hardware needed for implementation, and comments upon class of problems handled. Fig. 5.9 shows a plot of words to be retrieved for a memory with a five bit tag field in each word, corresponding to a memory size of 32 words. It is seen that our proposed scheme is equal to or better than all of the presented schemes in terms of speed, and in terms of the number of cycles needed to retrieve a word from memory. In terms of the absolute speed, the Foster method is somewhat faster in terms of gate delays per retrieval since it uses an external priority logic tree. The Foster scheme, however, is not useful as a tool for ordered retrieval, but only for multiple response resolution. At two retrievals per memory cycle, our proposed scheme is by far the fastest ordered retrieval scheme, even faster than the Miiller scheme which does not even produce ordering, only resolution. As far as the complexity of the hardware goes, our scheme is well within the realizable realm of LSI technology and in fact is no more complex than that used by Seeber and Lindquist or Yang and Yau [YAN66] in their implementation of Lewin's algorithm. We conclude that such a design as we have proposed here may be a useful and realizable tool for associatibe processing in any applications where ordered retrieval is important. One of the applications is to use it as a multiple match resolver as we have described in section 5.2.8. Another application is to use it as a file processor in data base applications. In the next section, we look at some of the requirements for offloading the processing onto a data base machine and see how the associative memory proposed in this section can be extended to sequential memories.

5.3 *DATA BASE MACHINES*

5.3.1 *Introduction*

A *Data Base Machine (DBM)* is defined as an architectural approach which raises the level of the interface from the CPU to the storage subsystem, and

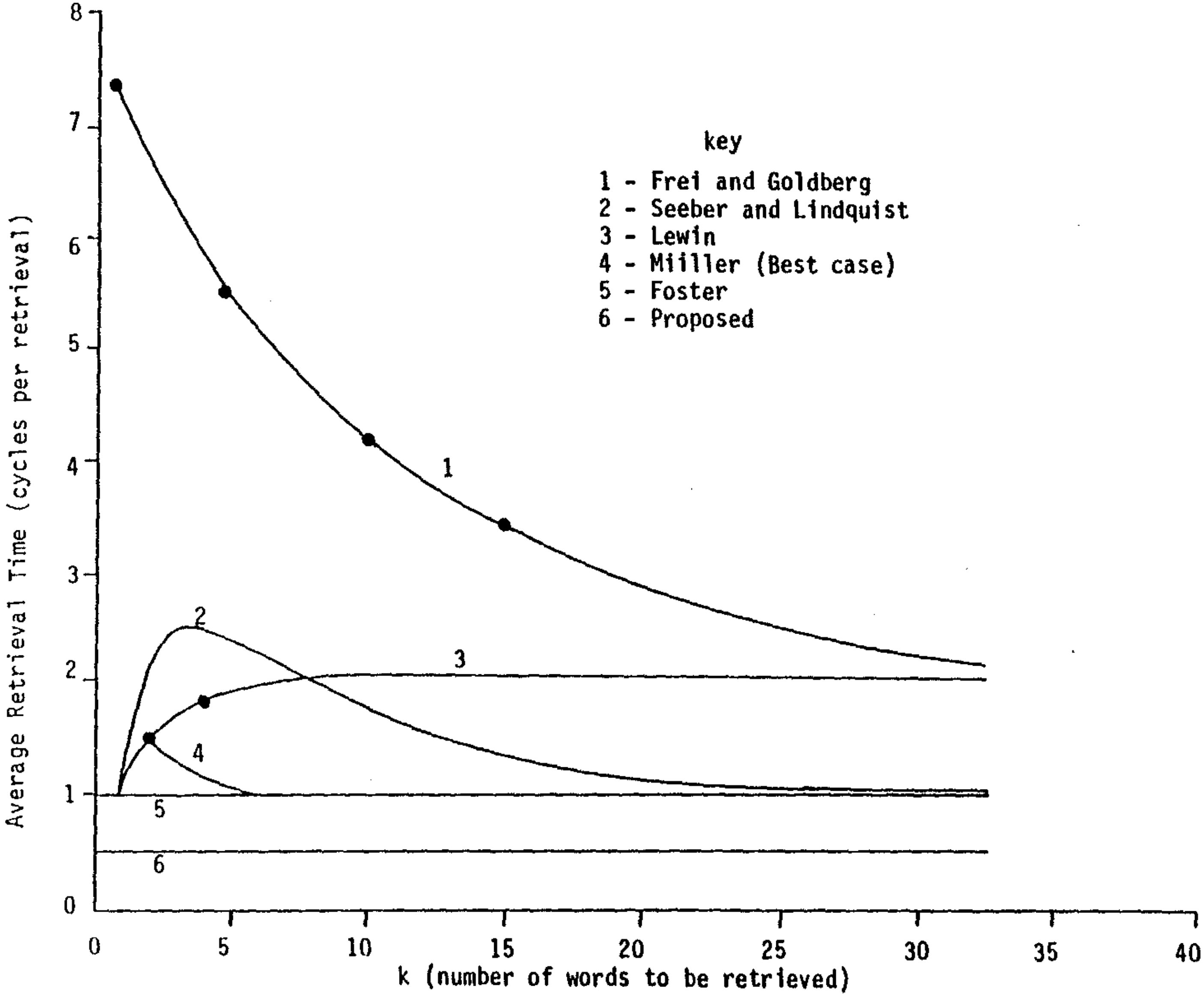


Figure 5.9 Comparison of Retrieval Speeds for a 5 bit Tag with k Words Flagged

distributes processing power closer to the devices on which data are stored [LAN79]. There have been many DBM designs, among them are Data Base Computer (DBC) [BAU76, KER79, BAN79], Context Addressed Segment Sequential Storage (CASSM) [LIP78, SU 79], Relational Associative Processor (RAP) [OZK77, SCH79], Rotating Associative Memory for Relational Data Base Applications (RARES) [LIN76], Datacomputer [MAR75], List Merging Network [HOL79], etc. Although most of these designs are directed towards a specific application, e.g. text processing, relational data bases, etc., the trend in the future is to utilize the available LSI technologies to design a more general purpose DBM. There have been many factors, both in the past and in the future, that pertain to the growth of DBM's. Apart from the growth of semi-conductor technologies and the rising need for larger data bases (Figures 1.2, 1.3, 1.4, Table 1.1), the most important factor that leads to the increasing hardware implementation of data base functions is the growth in complexity and size of data base management software. Because it is necessary to provide a high level view of the data to the users, it is essential to provide a complex translation from the physical data structure to the logical data view and vice versa. Conventionally, this has been done by the data base management software. Depending on how complex the translation mechanism is, the amount of software to be developed and the amount of execution time needed is also different. As an example, the INGRES data base takes 350,000 machine instructions to process a simple transaction which can be a retrieval or an update of a simple record of data. Out of these 350,000 instructions, only 25,000 instructions are real work that performs the actual function of the query. The other part of the work (325,000 instructions) is purely overhead which includes 25,000 instructions for parsing, 75,000 instructions for validity checks, 125,000 instructions for task switches and pipes and 100,000 instructions to interface with the users. Some of these overheads can be made smaller, e.g. the amount of validity checks can be reduced if the main memory is large enough and the system catalog can be put there; the user interface can be made less complex; the query can be parsed at compile time in order to eliminate the run time interpretor overhead, etc. The execution of a simple transaction is therefore CPU bound. On the other hand, in order to process a complex transaction in INGRES, which retrieves or updates multiple records, it takes about 25,000 machine instructions to process a 512 byte page and 20 msec. to fetch a page from the secondary storage. Out of these 25,000 instructions, only 6,000 are real work, the other part are overheads. However, the processing of a complex transaction can be speeded up by (i) compiling the query before execution; (ii) enlarging page size and/or adding drives; (iii) developing better decomposition strategies and (iv) building a one process real time system. As a result of these overheads, it is seen that the use of a DBM, which executes the query outside the CPU, reduces the execution overhead of the CPU and the I/O overhead in transferring data into the main memory.

5.3.2 *Issues in the Design of DBM's*

Traditionally, the design of DBM's are plagued by many issues. Among them are:

(1) *Parallelism - Kind and Degree*

The designer has to decide on the kind of functions that can be processed in parallel and in what degree. These functions include address mapping operators and the DB functions as well. As an example, the query processing in INGRES can be divided into four levels, (a) query modification which parses the query and reduces it to a useable form; (b) query decomposition which decomposes a query that accesses multiple files into multiple sub-queries that access single files; (c) one variable query processing which processes these sub-queries that access single files and (d) access method which translates the requests into physical disk accesses. We can have four different ways to cut the software into two sets so that one set resides in the CPU and the other set resides in the DBM. The analysis reduces to the allocation of processes in a two processor system and the max-flow min-cut network flow technique developed by Stone [STO77a] can be applied here. The parameters that the designers must consider include the speed of the DBM and the degree of parallelism needed. Further, they must consider the efficient scheduling of tasks on these processors.

(2) *Technology dependence*

The designer of the DBM must take into account the available technology. Further, the design must be able to evolve as new technologies are made available. Using disk technologies, there is a large overhead in translating the signals available from a disk head to a useable form by the DBM. With the availability of bubble and CCD memories, very little signal translation is necessary and the logic and the memory cells can be implemented together on the same chip.

(3) *Interface, where and in what form:*

The problem is to design a good interface between the DBM and the host processor. This interface may be implemented in hardware/firmware or software or a combinations of both. This interface translates queries from the host processor to DB functions processable by the DBM. Important questions like where to put this interface and how much capabilities it should have, must be answered. Should it be a part of the host, or should it be a part of the DBM? Should the interface be able to access the memory hierarchy? How should the interconnection network be between the DBM and the storage sub-system? What type of language primitives should be used? These questions have to be considered carefully by the designer.

(4) *Storage structure*

The kind of storage structure is very important. If keyed accesses, that is, accessing data via a key, are allowed, then additional hardware capabilities like associative memory or extra pointers are necessary to support it. Further, questions like whether the storage structure is dynamic should also be considered.

(5) *Backend primitives*

The designer has to trade the availability of backend primitives (which include functions like sorting, file merging, etc.) with the cost and the difficulty of implementing it.

(6) *Control algorithms*

Because the memories of a DBM are usually slow (of the crder of 100 μsec access time), much overlap and parallelism are necessary in order to achieve a high throughput. Control algorithms like scheduling and file placement and migration algorithms are therefore very important.

The designer of a DBM must consider all these issues together and make a judicious tradeoff in the design.

5.3.3 *Classification of DBM's*

The DBM proposed so far can be divided into two types, (1) backend systems using conventional mini-computers and (2) intelligent controllers which include cellular logic, associative memory and MIMD architecture. We describe each of them briefly here.

(1) *Backend systems using conventional mini- computers* (Figure 5.10)

In this design, backend systems are added to a generally large CPU in order to enhance its DB processing capabilities. The functions of the backend system can include access validation, storage management, concurrency control and

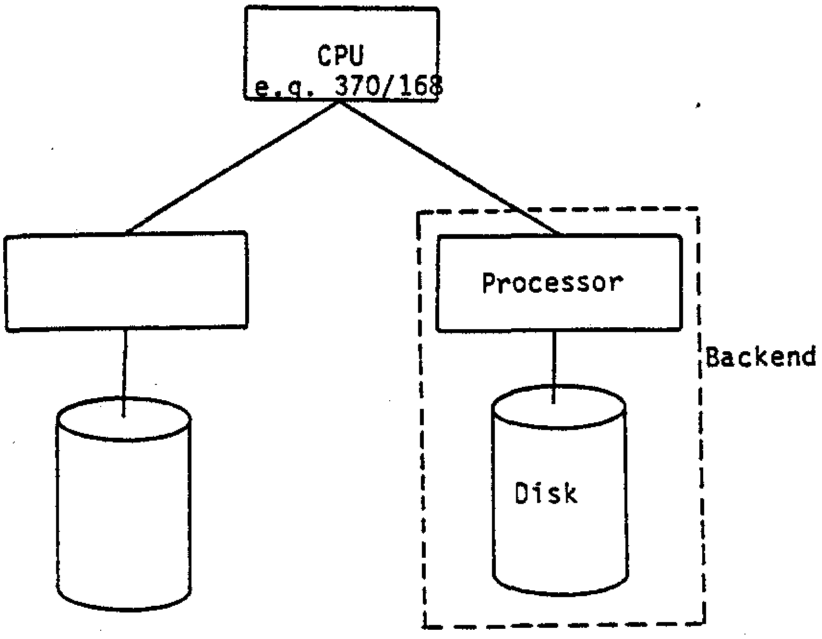


Figure 5.10 Backend Systems Using Conventional Mini-computers

I/O control. The advantages of such a system are that it allows concurrent sharing of a single data base and it provides better security, integrity and recovery measures because the backend machines provide a single gateway to the physical data base. In such a system, network protocols are designed so that the CPU can offload the processing onto the backend machines. As an example, in an IBM system, the CPU can be an IBM 370/168, and the network protocol is the SNA network protocol. In this case, it takes 10,000 to 30,000 machine instructions to execute the protocol and to offload the processing. However, if an INGRES data base is implemented on the system. and the system can only offload a fraction of the processing workload, e.g. validity checks cannot be offloaded, the speed improvement is only minimal. Further, there is an upper bound on the number of backend processors so that enough work can be offloaded onto these machines. Other disadvantages include costly software development and low reliability. The use of backend machines is therefore a temporary method to extend the processing power of a large CPU.

(2) *Intelligent controllers*

The use of intelligent controllers is an extension on the concept of backend machines. In the case of the backend machines, each one of them can control a set of disks and can perform high level data manipulations on the stored data. In the case of the intelligent controllers, the logic is partitioned further down onto the stored data. The characteristics of this design are that simpler, less costly designs are used and each of the controllers is dedicated to a smaller block of data. There may be a higher level controller which controls the intelligent controllers collectively. This design therefore approaches a multi-level control scheme. Basically, this design can be divided into three categories:

(a) *Cellular Logic* (Figure 5.11)

In this design, the processors are duplicated across each of the memory

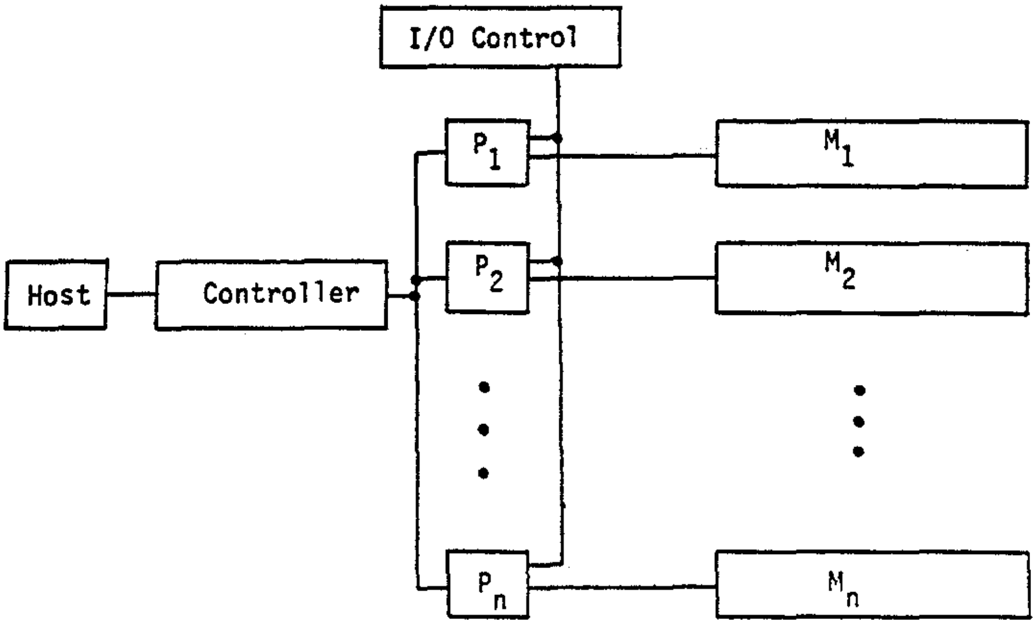


Figure 5.11 Intelligent Controllers - Cellular Logic

elements which may be a track of a disk. They provide associative search for data in the memory and they access data directly by value. Most of the conventional designs follow this principle, e.g. TapeDRUM [HOL56], Slotnick's Logic per Track Disk [SLO70], RAPID [PAR72], CASSM [LIP78, SU 79], RAP [OZK77, SCH79], RARES [LIN76], DBC [BAU76, KER79, BAN79], Chang's Major/Minor Loop Machine [CHA78], etc. Because the logics are distributed across the data, this design provides very fast searches and it reduces the software overhead by performing content addressing. Further, the architecture is very suitable for a relational data model which is a two dimensional data model. A relation can be placed so that all the tuples pass out in a bit-serial fashion to the cellular logic simultaneously. Other data models can be modified to fit the architecture by adding additional data structures, e.g. CASSM. However, there exists many disadvantages with this design: (i) Because of the large degrees of replication, the logic are bound to be simple. Usually, only simple functions like equality match, maximum search, etc., are implemented and the designs are directed towards specialized applications. (ii) The data base workload must be large (>40%) in order to keep the parallel resources utilized. (iii) In a large data base, the degree of replication may be large and the cost may be prohibitive. (iv) Because of the way that data is placed in the architecture, data types are limited to character strings and integers. More complex data structures would require more complicated external control. (v) If the DBM is built on a disk, the processors must be extremely fast because very fast signal translations are needed in order to process the disk data in real time. (vi) Lastly, I/O is usually the bottleneck. Although the processing can be done in parallel, I/O is usually done serially. However, it is hoped that the pre-processing using the DBM's has eliminated a significant portion of the data transfer.

(2) *Associative Memories* (Figure 5.12)

In this design, an associative memory, such as STARAN [GOO75] or the proposed design in Section 5.2 of this thesis, is used to provide associative search capabilities. The model in Figure 5.12 resembles a conventional memory hierarchy in which the fastest memory (the associative memory) is small and is interfaced to the slow mass storage through an intermediate buffer memory. The advantages of this design are rapid search for array resident data and its suitability for the relational data model. However, associative memories are still relatively expensive and large associative memories are not feasible. This design therefore experiences the usual problems of a memory hierarchy, namely, the swapping of the data across various levels of the hierarchy. It is still unknown whether the locality of data accesses in data bases is better than the locality of accesses in caches and virtual memory and is highly dependent on applications. Further, the technique is not effective for non-resident data and a high bandwidth bus is necessary to transfer data between the associative memory and the mass storage. In one such design [BER79] in which STARAN is used as the associative memory, it requires 1024 I/O lines with 300 to 450 nsec transfer time per bit slice to interconnect the associative memory with the buffer memory in order for the technique to be effective. This technique is therefore unduly expensive in the associative memory and the I/O bus.

(c) *Multiple- Instruction- Multiple- Data- Stream (MIMD) Architecture* (Figure 5.13)

In the MIMD architecture, the cellular logic have been pushed out of the memory elements and are interconnected with the memory elements through an interconnection network. This design offers more flexibility and better load balancing and allows the processors to be shared among the

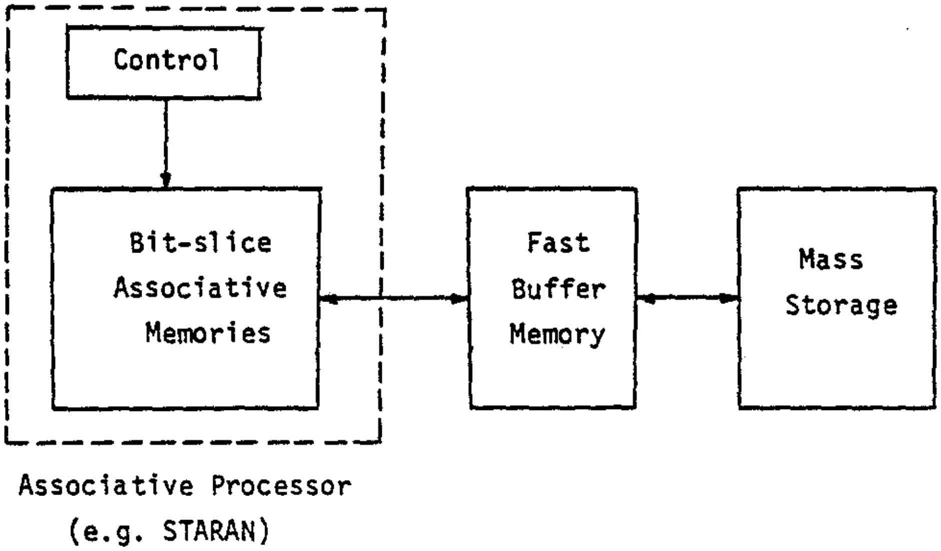
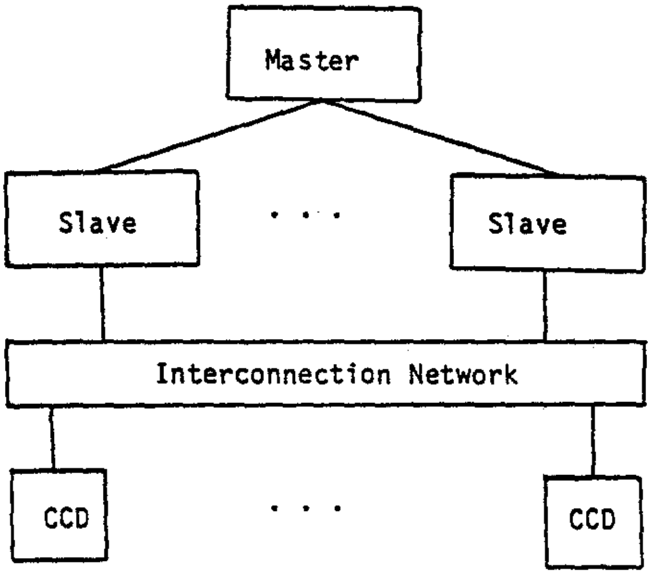

Control
Bit-slice Associative Memories
Fast Buffer Memory
Mass Storage
Associative Processor (e.g. STARAN)

Figure 5.12 Intelligent Controllers - Associative Memories



E.g. DIRECT

Figure 5.13 Intelligent Controllers - MIMD Architecture

memory elements. Because of the fact that each processor can access multiple memory elements simultaneously, it is easier to perform data base operations which require multiple files to be coupled, e.g. a multi-relation join. Further, expansion is easy and modular. However, this design suffers from the same disadvantages as the associative memory when the size of the memory is not large enough in which case excessive swapping will occur. Nonetheless, by using memory modules which are sufficiently large, it is possible that the amount of swapping can be limited to a tolerable amount. This design is exemplified by the DIRECT system [DEW79].

Since the DBM is a very special purpose hardware and requires a large degree of replication, it is important that unessential software are not placed in the DBM. In particular, software for protection, file system management, code swapping, task switches, pipes or system calls should be eliminated from the DBM. These software modules can be shared at a higher level with no adverse effects on the system performance. On the other hand, the DBM should have a thin collection of utilities, the run time DB management system and a self-managed buffer pool. The management of buffers is relatively easy here because the accesses are usually made in a sequential order.

In the next section, we present the design of a simple data base processor which is extended from the design of the associative memory presented earlier. Our design is totally hardware oriented and follows the same principle as the cellular logic approach. However, our design differs from the other designs in several features, (i) it is completely hardware controlled and therefore is very fast, (ii) the logic is very simple and therefore can be replicated easily and implemented on the same chip as the memory elements. The design is capable of equality, threshold, proximity and extremum searches.

5.3.4 *Extension of the Associative Memory Design to Sequential Memories*

Our design presented in Section 5.2 can be extended to the design of associative sequential memories which is made up of multiple loops of circulating bits shifting in synchronism. There is a read/write head for each loop so that one bit from each loop can be read or modified in one clock period. This can be extended to include multiple heads for each loop. Examples of such sequential memories include charge-coupled device memory, bubble memory and fixed head disk.

Since only one bit is available from each loop at any time, we can design the associative logic outside the sequential memory as shown in Figure 5.14. In this design, m words are stored in the memory, with one word occupying each loop. During a clock period, a bit-slice of these m words is shifted out of the memory. This bit-slice is then processed by the associative logic and the enable signals are stored in temporary flip flops. Note that in the design presented earlier, the enable signals propogate from the MSB to the LSB and the data are stored in flip-flops. In the case of a sequential memory, the enable signals are stored in temporary flip flops. As the bit-slice is shifted out, MSB first, the bit-slice, together with the stored enable signals, generate a new set of enable signals which are stored back into the flip-flops. The exact design is shown in Figure 5.15.

There are two advantages with this design. First, the additional logic for each word is very small and therefore the cost increase is minimal. Second, when the memory size is extended, only 8 lines due to the associative logic are needed to be connected between adjacent modules. Therefore, the memory size can be modularly expanded. Moreover, the amount of bit-slice control logic is small, so we can design a memory with n modules, each with its own associative and bit-slice control logic. During normal operations, each module can perform independent associative search operations. When it is necessary to perform associative search operations on 2 or more adjacent modules, all except one of

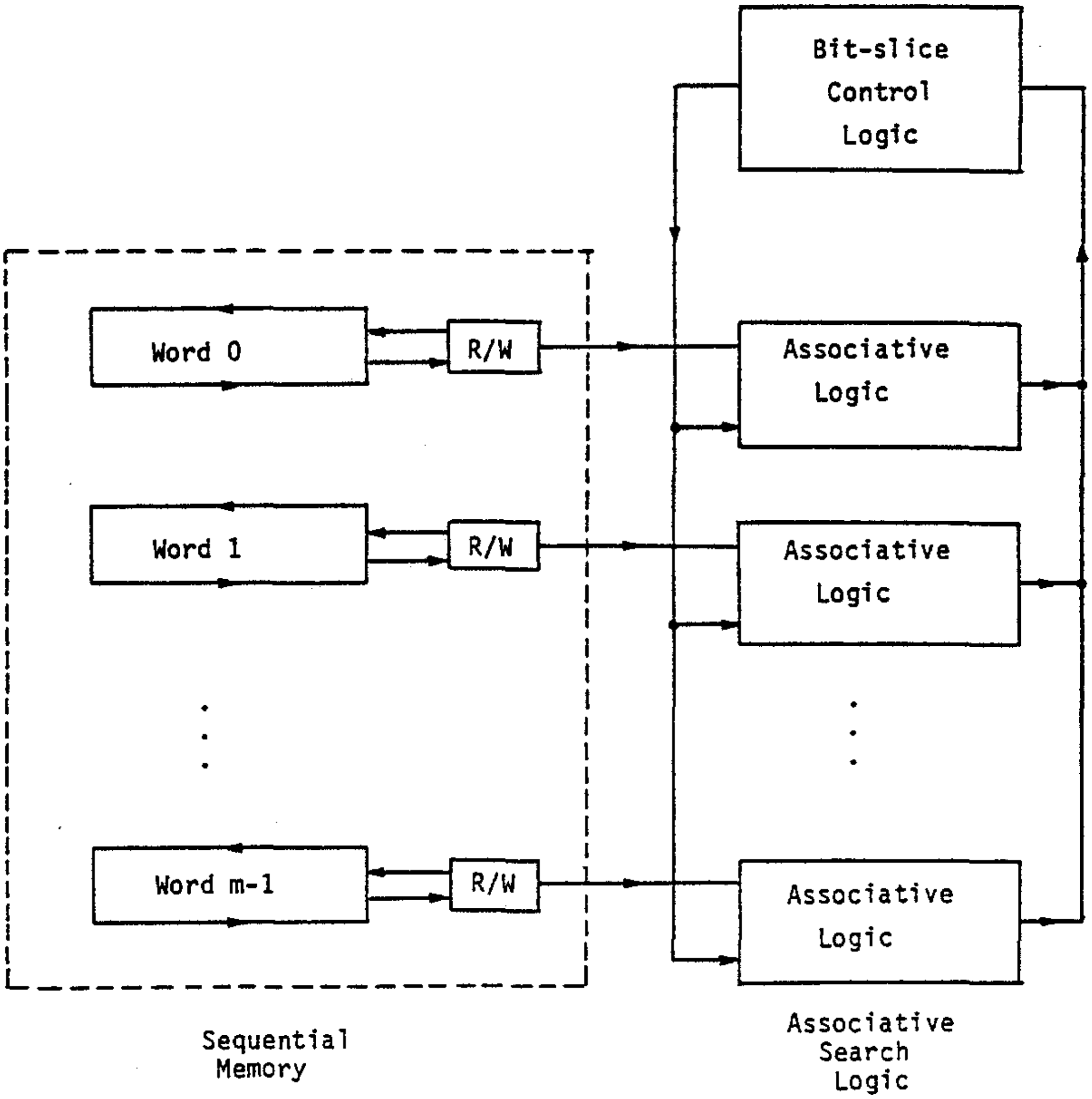


Figure 5.14 Associative Sequential Memory

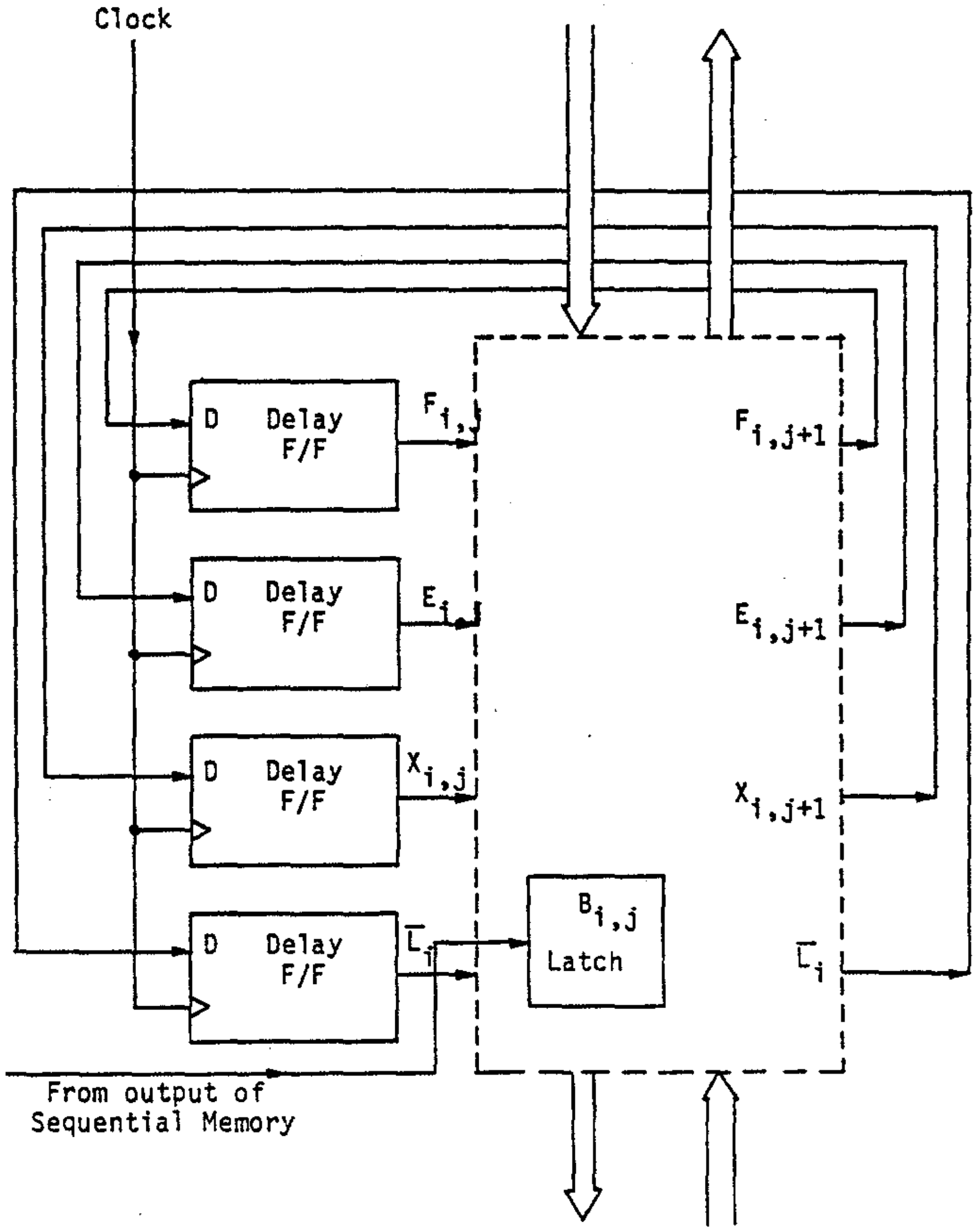


Figure 5.15 Associative Logic for Associative Sequential Memory

the bit-slice control logic for these modules are switched out of the system and the feedback lines are connected together to form a large block of associative memory. This dynamic reconfiguration capability is useful in applications where the nature and the size requirements may change dynamically. However, there are two limitations with this design. First, the words must be organized as described here because our design can only process one bit-slice at a time. Second, it is limited to memory types in which these logic can be easily implemented in LSI technology, e.g. CCD memory and bubble memory. In disk technology, the associative logic have to be implemented on a separate chip and the amount of interconnections between the memory and the associative logic may become prohibitively large. In a feasible implementation, e.g. CCD memory, this design can be used as the lowest level of a DBM. Higher level control may be designed to include more complicated functions.

5.4 *CONCLUSION*

We have presented in this chapter two hardware features to support data management on a distributed data base. The first design is an associative memory which is capable of equality, threshold, proximity and extremum searches. The design is completely asynchronous and is bit-serial and word-parallel, that is, the enable signals propogate from bit-slice to bit-slice, but all the processing within a bit slice is done in parallel. The propogation time across a bit-slice is 1 to 7 gate delays and each cell has a complexity of 17 gates. This design is by far the fastest in the literature. Although memory expansion presents a slight problem, but by extending the dimensions of the memory, the memory can be expanded with only a slight degradation in performance.

The second hardware design presented is an extension of the associative memory to data base machines. It is shown that the concept of the bit-serial associative memory can be extended to associative sequential memory. The design is useful in upgrading the capabilities of the mass storage and reducing the amount of data transfers across different levels of the memory hierarchy.

6. *CONCLUSION*

In this thesis, the issues on the resource management of data on a distributed data base (DDB) system are studied. These issues are concerned with the management of data and files as resources so that they can be shared efficiently by the users. The major issues studied are:

(1) *Query Decomposition on DDB's*

A query is an access request made by a user or a program in which one or more files have to be accessed. When multiple files are accessed by the same query on a DDB, these files usually have to reside at a single location before the query can be processed. Substantial communication overhead may be involved if these files are geographically distributed. It is therefore necessary to decompose the query into sub-queries so that each sub-query accesses a single file. These sub-queries may then be processed in parallel at any location which has a copy of the required file. The results after the processing are sent back to the requesting location. It is generally true that the amount of communications needed to transmit the results is much smaller than the amount needed to transfer the files. This approach has been proposed in the design of the centralized version of INGRES and is extended to the design of SDD-1, a distributed data base. However, in some cases, decomposition is impossible and some file transfers are still necessary. In order to avoid these extra transfers, two cost reduction models have been designed to reduce the operational costs of a relational data base. The first model reduces the retrieval cost but increases the update cost by adding redundant information to each domain of a relational data base so that relational operations such as joins and aggregate functions can be performed without any file transfers. The second model reduces the update cost but increases the retrieval cost by partitioning the relations into segments so that they can be updated more readily. These two cost reduction models can be combined to form a unified approach to reduce the operational costs of DDB's. Further, it is shown that the optimization of placements of multiple relations under the use of these techniques can be done independently for each relation.

(2) *File Placement and Migration*

This issue relates to the distribution and migration of data base components, namely, schema, data and control programs on the DDB with the objective of minimizing the overall storage, migration, updating and operational costs on the system. In this thesis, the problem of file placements and the problem of selecting the times for migration under changing access frequencies have been proved to be NP-complete. Further, the isomorphism between the file placement problem and the facility location problem are shown. The implications of the last result are two folds. First, many results which have been derived in one problem can now be applied to solve the other problem. Second, some results obtained earlier for one problem can be shown to be weaker than the corresponding results derived for the other problem. A file placement heuristic is developed. While not necessary yielding optimal design, the heuristic yields solutions of lower cost than those generated by other currently available heuristics.

(3) *Task Scheduling*

In task scheduling, the requests on the nodal computer system and the distributed computer system are sequenced so that high parallelism and overlap can be achieved. The requests may be a single word fetch or it may be a page or file access. A model for the scheduling of tasks on a distributed system has been developed. This model assumes that global control is infeasible and all the scheduling decisions have to be made locally at each node. It is shown that the scheduling of tasks in this model, when all the task processing times are deterministic, is NP-complete. A heuristic has been developed and the performance of this heuristic has been verified using simulations. A more restricted model, which represents an organization of an interleaved memory system, is also proposed. By using the additional constraints, it is proved that the optimal scheduling problem is polynomially solvable. The performance of the scheduling algorithm has been verified using simulations. Further, the degradation in performance due to dependencies has also been estimated.

(4) *Hardware support*

Beyond the problem of resource management studied, the hardware support for the data base systems has also been investigated. In particular, an associative memory which is capable of equality, threshold and extremum searches in a time independent of the number of words in the memory has been designed. The complexity of the design is 17 gates/cell. The design is asynchronous and utilizes a word-parallel and bit-serial algorithm. The delay is 1 to 4 gate delays across each bit-slice. This design can be applied to the resolution of multiple responses. Further, such a design is not restricted to associative memories and can be applied to the design of associative sequential memories and data base machines.

APPENDIX A THE ISOMORPHISM BETWEEN STONE'S PROCESS ALLOCATION PROBLEM AND THE SINGLE COMMODITY QUADRATIC ASSIGNMENT PROBLEM

Stone's process allocation problem studies the allocation of processes to computers [STO77a, STO77b, STO78a, STO78b, STO79]. The amount of communications between two processes are defined and this in turn defines the cost to be incurred if these two processes run on different computers. There is also a cost of executing a process on a computer. The problem is to place the processes so that the total cost of the system is minimum.

On the other hand, the single commodity quadratic assignment problem studies the allocation of plants to plant sites. There are certain fixed quantities of the single type of commodity that are to be shipped between the plants and these define an overhead cost to the system if these plants are located in different plant sites. There are also fixed costs of locating a plant at a plant site. The problem is to locate the plants so that the total cost is minimum.

We can now prove the following theorem.

THEOREM A- 1

Stone's process allocation problem is isomorphic to the single commodity quadratic assignment problem.

Proof

The theorem can be proved by associating the variables of Stone's problem with the variables of the single commodity quadratic assignment problem. This association is shown in Table A-1.

Table A- 1 Mapping between Stone's Process Allocation Problem and the Single Commodity Quadratic Assignment Problem

Stone's Process Allocation Problem	Single Commodity Quadratic Assignment Problem
Locations of computers	Possible plant sites
Process	Plant
Communications between two processes	Commodity to be shipped between two plants
Cost of communication between two computers	Cost to ship commodity between two plant sites
Fixed cost of executing a process on a computer	Fixed cost of locating a plant at a plant site

Q.E.D.

APPENDIX B THE LINEAR PROGRAMMING LOWER BOUND OF A CANDIDATE PROBLEM [*EFR66*]

Efroymson and Ray's formulation of the linear programming lower bound is based on the optimization problem of Eq. 3.1, with an exception that $\sum_{k=1}^{n} X_{j,k} S_{j,k}$ is not evaluated to be $\min_{k \in I} S_{j,k}$ where $X_{j,k}$ is the fraction of Q_j that is directed towards node k. The optimization problem that Efroymson and Ray considered is (using the notations defined in this thesis):

$$\min C(I) = \sum_{j,k} Q_j S_{j,k} X_{j,k} + \sum_{k} G_k Y_k \tag{B-1}$$

such that

$$1 = \sum_{k=1}^{n} X_{j,k} \qquad (j=1, \ldots, n)$$

$$0 \leqq X_{j,k} \leqq Y_k \leqq 1 \qquad (i,j=1, \ldots, n)$$

$$Y_k = 0, 1$$

By defining the following notations,

N_j = set of indexes of those nodes that can be accessed by user j;

P_k = set of indexes of those users that can access node k;

n_k = number of elements in P_k.

The objective function can be rewritten as:

min

$$C(I) = \sum_{j,k} Q_j S_{j,k} X_{j,k} + \sum_{k} G_k Y_k \tag{B-2}$$

such that

$$1 = \sum_{k \in N_j} X_{j,k} \qquad (j=1, \ldots, n)$$

$$0 \leqq \sum_{j \in P_k} X_{j,k} \leqq n_k Y_k \qquad (k=1, \ldots, n)$$

$$Y_k = 0, 1$$

Recall that,

$K_0 = \{k : Y_k = 0\};$

$K_1 = \{k : Y_k = 1\};$

$K_2 = \{k : Y_k = \text{unassigned}\}.$

The linear programming solution to the above optimization problem, neglecting the integrality constraint of Y_k, is,

$$X_{j,k} = \begin{cases} 1 & \text{if } S_{j,k} + \frac{g_k}{n_k} = \min_{l \in K_1 \cup K_2} \left[S_{j,l} + \frac{g_l}{n_l} \right] \\ 0 & \textit{otherwise} \end{cases} \tag{B-3}$$

$$Y_k = \left[\frac{1}{n_k} \right] \sum_{j \in P_k} X_{j,k} \tag{B-4}$$

where

$$g_k = \begin{cases} G_k & k \in K_2 \\ 0 & k \in K_1 \end{cases}$$

This is the optimal solution because for $k \in K_2$,

$$\sum_{j \in P_k} X_{j,k} \leq n_k Y_k$$

which implies that in the optimal solution, the equality sign will hold, i.e.,

$$\sum_{j \in P_k} X_{j,k} = n_k Y_k$$

or

$$\frac{1}{n_k} \sum_{j \in P_k} X_{j,k} = Y_k$$

Substituting this value for Y_k, $k \in K_2$ into the objective function, the linear program becomes,

$$\min C(I) = \sum_{k \in K_1} G_k + \min \left\{ \sum_{k \in K_1} Q_j S_{j,k} X_{j,k} + \sum_{k \in K_2} \left[Q_j S_{j,k} + \frac{G_k}{n_k} \right] X_{j,k} \right\}$$

such that,

$$1 = \sum_{k \in N_j} X_{j,k} \qquad (j = 1, \ldots, n)$$

This lead to the optimal solution.

APPENDIX C THE EXPECTED VALUE OF A CANDIDATE PROBLEM

Recall that,

$$K_0 = \{j: Y_j = 0\}$$

$$K_1 = \{j: Y_j = 1\}$$

$$K_2 = \{j: Y_j = unassigned\}$$

We can rewrite the objective function (Eq. 3.1) on condition on K_0 and K_1.

$$\begin{aligned} C(I) = & \sum_{i \in K_1} G_i \\ & + \sum_{i \in K_0} Q_i * \min_{j \in I} S_{i,j} \\ & + \sum_{i \in K_2} Q_i * \min_{j \in I} S_{i,j} + \sum_{i \in K_2} G_i Y_i \end{aligned}$$

$$C(I) = \sum_{i \in K_1} G_i + \sum_{i \in K_0 \cup K_2} Q_i * \min_{j \in I} S_{i,j} + \sum_{i \in K_2} G_i Y_i \qquad \text{(C-1)}$$

where G_i is defined in Eq. 3.2.

Let

$$Z_1 = \sum_{i \in K_0 \cup K_2} Q_i * \min_{j \in I} S_{i,j} \qquad \text{(C-2)}$$

$$Z_2 = \sum_{i \in K_2} G_i Y_i \qquad \text{(C-3)}$$

So

$$C(I) = \sum_{i \in K_1} G_i + Z_1 + Z_2 \qquad \text{(C-4)}$$

Assuming that each of the combinations of Y_j for $j \in K_2$ can be assigned uniformly, we would like to find the expected value of C(I). We first define some notations:

For each row i of matrix S, we define a mapping μ_i such that

$$\mu_i : j \to k \qquad j, k \in \{1, \ldots, n\} \; such \; that \; S_{i,\mu_i^{-1}(k)} \leqq S_{i,\mu_i^{-1}(k+1)}$$

The mapping μ_i maps the original set of nodes onto a new set such that the costs of access from node i in the mapped matrix are in increasing order.

$$S_{i,t} = \min_{j \in K_1} S_{i,j}$$

$t \in K_1$ is the node which has the minimum cost of access from node i.

$$|\overline{K_2}| = |K_0 \cup K_1| \quad (\textit{cardinaltiy of } \overline{K_2})$$

$$K = K_0 \cup K_1 \cup K_2$$

$$C(K) = \begin{cases} 2^{n-|\overline{K_2}|}-1 & \text{if } |K_1| = 0 \\ 2^{n-|\overline{K_2}|} & \text{if } |K_1| > 0 \end{cases}$$

$$K_{2iq} = \{x : x \in K_2 \text{ and } \mu_i(x) \geq \mu_i(q)\}$$

Now

$$E(Z) = \sum_{i \in K_1} G_i + E(Z_1) + E(Z_2)$$

$$E(Z_1) = E\Big(\sum_{i \in K_0 \cup K_2} Q_i * \min_{j \in I} S_{i,j}\Big)$$

$$= \sum_{i \in K_0 \cup K_2} Q_i * E\big(\min_{j \in I} S_{i,j}\big)$$

$$E(\min_{j \in I} S_{i,j}) = \frac{\left[\displaystyle\sum_{\substack{q \in K_2 \\ \mu_i(q) < \mu_i(t)}} S_{i,q} 2^{(|K_{2iq}|-1)} + S_{i,t} 2^{|K_{2it}|}\right]}{C(K)}$$

$$E(Z_2) = E\Big(\sum_{j \in K_2} G_j Y_j\Big)$$

$$= \sum_{j \in K_2} G_j E(Y_j)$$

$$E(Y_i) = \frac{2^{(n-|\overline{K_2}|-1)}}{C(K)}$$

$$E(Z) = \sum_{i \in K_1} G_i \tag{C-5}$$

$$+ \frac{1}{C(K)} \sum_{i \in K_0 \cup K_2} Q_i * \left[\sum_{\substack{q \in K_2 \\ \mu_i(q) < \mu_i(t)}} S_{i,q} 2^{(|K_{2iq}|-1)} + S_{i,t} 2^{|K_{2it}|}\right]$$

$$+ \Big(\sum_{i \in K_2} G_i\Big) \frac{2^{(n-|\overline{K_2}|-1)}}{C(K)}$$

APPENDIX D BIBLIOGRAPHY

[AKI77] Akinc, U., and Khumawala, B., "An Efficient Branch and Bound Algorithm for the Capacitated Warehouse Location Problem", *Management Science,* Vol. 23, No. 6, Feb. 1977, pp. 585-594.

[ALC76] Alcouffe, A., and Muratet, G., "Optimum Location of Plants", *Management Science,* Vol. 23, No. 3, Nov. 1976, pp. 267-274.

[AND67] Anderson, D.W., Sparacio, F.J. and Tomasulo, R.M., "The IBM System 360 Model 91: Machine Philosophy and Instruction Handling", *IBM J. of Research and Develop.*, Jan. 1967, pp.8-24.

[AND75] Anderson, G. A. and Jensen, E. D., "Computer Interconnection Structures: Taxonomy, Characteristics and Examples", *Computing Surveys,* Vol. 7, No. 4, December, 1975.

[ARM63] Armour, G. C., and Buffa, E. S., "A Heuristic Algorithm and Simulation Approach to Relative Location of Facilities", *Management Science,* Vol. 9, No. 2, Jan 1963, pp. 294-309.

[ASC74] Aschim, F., "Data-Base Networks - An Overview", *Management Information,* Vol. 3, No. 1, 1974.

[BAC75] Bachman, C., "Trends in Data Base Management", *Proc. of AFIPS National Computer Conference, 1975,* Vol. 44, AFIPS Press, Montvale, NJ, 1975, pp. 569-576.

[BAD78] Badal, D. Z., "Data Base System Integrity", *Digest of Papers*, Compcon Sp. 78, pp. 356-359.

[BAN79] Banerjee, J., Hsiao, D. K., and Kannon, K., "DBC - A Data Base Computer for Very Large Data Bases", *IEEE Trans. on Computers,* Vol. C-28, No. 6, June 1979, pp. 414-429.

[BAS70] Baskett, F., Browne, J. C., and Raike, W. M., "The Management of a Multi-level Non-paged Memory System", *Spring Joint Computer Conference,* 1970, pp. 459-465.

[BAS75] Baskett, F., Chandy, K. M., Muntz, R. R., and Palacios, F. G., "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers", *JACM,* Vol. 22, No. 2, April, 1975, pp. 248-260.

[BAS76] Baskett, F., and Smith, A. J., "Interference in Multiprocessor Computer Systems with Interleaved Memory", *CACM,* Vol 19, No. 6, June 1976, pp. 327 - 334.

[BAU58] Baumol, W. J., and Wolfe, P., "A Warehouse Location Problem", *Operations Research,* Vol. 6, March-April, 1958, pp. 252-263.

[BAU76] Baum, R. I., Hsiao, D. K., "Data Base Computers - A Step Towards Data Utilities", *IEEE Trans. on Comp.*, Vol. C-25, No. 12, Dec. 1976.

[BEL77] Belady, L. A., and Lehman, M. M., *The Characteristics of Large Systems,* IBM Research Report, RC6785, Sept. 1977.

[BEN77] Bentley, J. L., and Shamos, M. I., *Divide and Conquer for Linear Expected Time,* Dept. of Computer Science and Mathematics Report, Carnegie-Mellon University, 1977.

[BER79] Berra, P. B., and Oliver, E., "The Role of Associative Array Processors in Data Base Machine Architecture", *IEEE Computer,* March 1979, pp. 53-61.

[BHA75] Bhandarkar, D. P., "Analysis of interference in Multiprocessors", *IEEE Trans. on Computers,* Vol. C-24, No. 9, Sept. 1975, pp. 897 - 908

[BOB71] Bobeck, A. H., and Scovil, H. E. D., "Magnetic Bubbles", *Scientific American*, Vol. 224, No. 6, pp. 78-90, June 1971.

[BOL67] Boland, L. J., Granito, G.D., Marcotte, A. V., Messina, B. V., and Smith, J. W., "The IBM System 1360 Model 91 : Storage Systems", *IBM J. of Res. and Dev.*, Jan. 1967, pp. 54 - 68.

[BON64] Bonner, R. E., "On some Clustering Techniques", *IBM J. of Research and Development*, Vol. 8, No. 1, Jan. 1964, pp. 22-32.

[BOO76] Booth, G. M., "Distributed Data Bases - Their Structure and Use", *Infotech State of the Art Report on Distributed Systems*, 1976.

[BRA76] Bray, O. H., "Distributed Data Base Design Considerations", *Trends and Applications, Computer Networks, 1976.*

[BRI77] Briggs, F. A. and Davidson, E. S. "Organization of Semiconductor Memories for Parallel - Pipelined Processors", *IEEE Trans. on Comp.*, Vol. C-26, No. 2, Feb. 1977, pp. 162 - 169.

[BUR70] Burnett, G. J., and Coffman, Jr. C. G., "A Study of Interleaved Memory Systems", *Proc. AFIPS 1970 SJCC*, Vol 36, pp. 467 -474, AFIPS Press, Montvale, N.J.

[BUR73] Burnett, G. J., and Coffman, Jr. E. G., "A Combinational Problem Related to Interleaved Memory Systems", *JACM*, 20, 1, Jan. 1973, pp. 39 - 45.

[BUR75] Burnett, G. J., and Coffman, Jr. E. G., "Analysis of Interleaved Memory Systems Using Bolckage Buffers", *CACM*, Vol. 18, No. 2, Feb. 1975, pp. 91 - 95.

[CAS72] Casey, R. G., "Allocation of Copies of a File in an Information Network", *AFIPS, SJCC*, 1972, pp. 617-625.

[CHA75] Chandy, K. M. and Herzog, U., and Woo, L., "Approximate Analysis of General Queeuing Networks", *IBM J. of Research and Development*, Jan 1975, pp. 43-49.

[CHA77] Chang, D. Y., Kuck, D. J., and Lawrie D. H., "On the Effective Bandwidth of Parallel Memories", *IEEE Trans. on Comp.*, May 1977, pp. 480 - 490.

[CHA78] Chang, H., "On Bubble Memories and Relational Data Base", *4th Int'l Conf. on Very Large Data Bases*, Berlin, Sept. 13-15, 1978, pp. 207-229.

[CHU69] Chu, W. W., "Multiple File Allocation in a Multiple Computer System", *IEEE Trans. on Comp.*, Vol. C-18, No. 10, Oct. 1969, pp. 885-889.

[CHU76] Chu, K. C., *Decentralized Dynamic Allocation Scheme for Large Congested Networks*, IBM Research Report, RC6337, 1976.

[COD70] Codd, E. F., "A Relational Model of Data for Large Shared Data Bases", *CACM*, Vol. 13, No. 6, June 1970.

[COF71] Coffman, Jr.,E. G., Burnett, G. J., and Snowdon, R. A., "On the Performance of Interleaved Memories with Multiple Word Bandwidths", *IEEE Trans. Comput.*, C-20, 12, Dec. 1971, pp. 1570 - 1573.

[DAN51] Dantzig, G. B., "Application of the Simplex Method to a Transportation Problem", Ch. 23 of *Activity Analysis of Production and Allocation*, T. C. Koopmans Ed., Cowles Commission Monograph, No. 13, John Wiley and Sons, 1951.

[DAT77] Date, C. J., *An Introduction to Data Base Systems*, 2nd Edition, Addison-Wesley, 1977.

[DDP78] Distributed Data Processing Workshop, Stanford University, Feb. 15-17, 1978.

[DEN70] Denning, P. J., "Virtual Memory", *Computing Surveys*, Vol. 2, No. 3, Sept. 1970, pp. 62-97.

[DEW79] DeWitt, D. J., "DIRECT - A Multi-processor Organization for Supporting Relational Data Base Management Systems", *IEEE Trans. on Computers*, Vol. C-28, No. 6, June 1979, pp. 395-406.

[DOW77] Downs, D., and Popek, G. J., "A Kernel Design for a Secure Data Base Management System", *Proc. Very Large Data Base*, Oct. 1977, pp. 507-514.

[DRA66] Draper, N. R. and Smith H., Applied Regression Analysis, John Wiley and Sons, New York, 1966.

[EFR66] Efroymson, M. A., and Ray, T. C., "A Branch and Bound Algorithm for Plant Location", *Operations Research*, May-June 1966, pp. 361-368.

[EPS78] Epstein, et. al., *Distributed Query Processing in a Relational Data Bse System*, Report No. UCB/ERL M78/18, Electronics Research Laboratory, University of California, Berkeley, 1978.

[ERL74] Erlenkotter, D., "Dynamic Facility Location and Simple Network Models" *Management Science Notes*, Vol. 26, No. 9, May 1974, pp. 1131.

[ESW74] Eswaran, K. P., "Placement of Records in a File and File Allocation in a Computer Network", *Information Processing, 74*, IFIPS, North Holland Publishing Co., 1974.

[ESW76] Eswaran, K. P. et. al., "The Notions of Consistency and Predicate Locks in a Data Base System", *CACM*, Vol. 19, No. 11, Nov. 1976, pp. 624-633.

[FEL50] Feller, W. *An Introduction to Probability Theory and its Applications*, Vol. I, John Wiley & Son Inc. 3rd Ed. 1950.

[FEL66] Feldman, E., Lehner, F. A., and Ray, T. L., "Warehoude Location under Continuous Economies of Scale", *Management Science*, Vol. 12, No. 9, May 1966, pp. 670-684.

[FEN74] Feng, T., "Data manipulating functions in parallel processor and their implementations", *IEEE Trans. Comput.*, Vol. C-23, pp. 309-318, Mar. 1974.

[FET76] Feth, G. C., "Memories: Smaller, Faster, and Cheaper", *IEEE Spectrum*, June, 1976, pp. 36-43.

[FLO64] Flores, I., "Derivation of a Waiting-Time Factor for a Multiple Bank Memory", *JACM*, Vol. 11, No. 3, July 1964, pp. 265 - 282.

[FLY66] Flynn, M. J., "Very High Speed Computing Systems", *Proc. of the IEEE*, Vol. 54, pp. 1901-1909

[FOS68] Foster, C. C., "Determination of priority in associative memories", *IEEE Trans. Electron. Comput.*, Vol. EC-17, pp. 788-789, Aug. 1968.

[FOS76] Foster, C. C., *Content Addressable Parrallel Processors*, New York: Van Nostrand Reinhold, 1976.

[FOS77] Foster, D. V., Dowdy, L. W., Ames, J. E. IV, "File Assignment in Star Network", *Proc. of the 1977 Sigmetrics/CMG VIII Conf. on Comp Perf.: Modelling, Measurement and Management*, Washington, D.C., Nov. 1977, pp. 247-254.

[FRA63] Francis, R. L., "A Note on the Optimum Location of New Machines in existing Plant Layouts", *The Journal of Industrial Engineering*, Jan-Feb 1963.

[FRE61] Frei, E. H. and Goldberg, J., "A method for resolving multiple responses in a parallel search file", *IRE Trans. Electron. Comput.*, Vol. EC-10, p.718, Dec. 1961.

[FRY76] Fry, J. P. and Sibley, E. H., "Evolution of Data Base Management Systems", *Computing Surveys*, Vol. 8, No. 1, March 1976, pp. 7-42.

[GIG73] Giglio, R. J., "A Note on the Deterministic Capacity Problem", *Management Science Notes*, Vol. 19, No. 12, Aug. 73, pp. 1096-1099.

[GEO72] Geoffrion, A. M. and Marsten, R. E., "Integer Programming: A Framework and State-of-the-Art Survey", *Management Science*, Vol. 18, No. 9, May, 1972, pp. 465-491.

[GHO76] Ghosh, S. P., "Distributing A Data Base with Logical Associations on a Computer Network for Parallel Searching", *IEEE Trans. on Software Engr.*, Vol. SE-2, No. 2, June, 1976, pp. 106-113.

[GIL70] Gilmore, P. C., "Optimal and Sub-optimal Algorithms for the Quadratic Assignment Problem", *Journal of the Society for Industrial and Applied Mathematics*, Vol. 10, No. 2, June, 1962, pp. 305-313.

[GOO75] Goodyear Aerospace Corporation, *STARAN Reference Manual*, Revision 2, GER-15636B, Akron, Ohio, June 1975.

[GRA70] Graves, G. W., and Whinston, A. B., "An Algorithm for the Quadratic Assignment Problem", *Management Science*, Vol. 16, No. 7, March 1970, pp. 453-471.

[GRA77a] Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G., "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey", *Proc. of Discrete Optimization, 1977*, Vancouver, Canada, Aug. 8-12, 1977.

[GRA77b] Grapa, E., Belford, G. G., "Some Theorems to Aid in Solving the File Allocation Problem", *CACM*, Vol. 20, No. 11, Nov. 1977, pp. 878-882.

[HAN66] Hanlon, A. G., "Content-addressable and associative memory systems: A survey", *IEEE Trans. Electron. Comput.*, Vol. EC-15, pp. 509-521, Aug. 1966.

[HEL67] Hellerman, H., *Digital System Principles*, McGraw Hill, New York, 1967, pp. 228 - 229.

[HEV79] Hevner, A. R., and Yao, S. B., "Query Processing in Distributed Data Bases", *IEEE Trans. on Software Engineering*, Vol. SE-5, No. 3, May 1979, pp. 177-187.

[HIL66a] Hilberg, W., "Simultaneous multiple response in associative memories and readout of the detector matrix", *IEEE Trans. Electron. Comput.*, Vol. EC-15, pp. 117-118, Feb. 1966.

[HIL66b] Hiller, F. S., and Connors, M. M., "Quadratic Assignment Problem Algorithms and the Location of Indivisible Facilities", *Management Science*, Vol. 13, No. 1, Sept. 1966, pp. 42-57.

[HOF78] Hofri, M., and Jenny, C. J., *On the Allocation of Processes in Distributed Computer Systems*, IBM Research Report, RZ905, 1978.

[HOL56] Hollander, G. L., "Quasi-Random Access Memory Systems", *AFIPS Conf. Proc. EJCC*, 1956, pp. 128-135.

[HOL79] Hollar, L. A., "A Design for a List Merging Network", *IEEE Trans. on Computers*, Vol. C-28, No. 6, June 1979, pp. 406-413.

[HOO77] Hoogendoorn, C.H., "A General Model for Memory Interference in Multi-processors", *IEEE Trans. on Comp.*, Vol. C-26, No. 10, Oct. 1977, pp.998-1005.

[HSI77] Hsiao, D. K., and Madnick, S. E., "Database Machine Architecture in the Context of Information Technology Evolution", *Proc. Very Large Data Base*, Oct. 1977, pp. 63-84.

[HUG75] Hughes, W. C., et. al., "A Semiconductor Nonvolatile Electron Beam Accessed Mass Memory", *Proc. IEEE*, Vol. 63, No. 8, Aug. 1975, pp. 1230-1240.

[JAR71] Jardine, N. and Van Rijsbergen, C. J., "The Use of Hierarchical Clustering In Information Retrieval", *Information Storage and Retrieval*, Vol. 7, 1971, pp. 225-239.

[JEN77] Jenny, C. J., *Process Partitioning in Distributed Systems*, IBM Research Report, RZ873, 1977.

[JOH54] Johnson, S. M., "Optimal Two- and Three Stage Production Schedule with Setup Times included", *Naval Research Logistics Quarterly*, Vol. 1, pp. 61-68.

[KAR72] Karp, R. M., "Reducibility among Combinatorial Problems", *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher eds., Plenum Press, New York, 1972, pp. 85-104.

[KAU69] Kautz, W. H., "Cellular logic in memory arrays", *IEEE Trans. Electron. Comput.*, Vol. EC-18, pp. 719-727, Aug. 1969.

[KAU71] Kautz, W. H., "An augmented context-addressed memory array for implementation with large-scale integration", *JACM*, Vol. 18, pp. 19-33, Jan. 1971.

[KEM65] Kemeny, J. G. and Snell, J. L., *Finite Markov Chains*, D. Van Nostrand Company, Inc. 1965.

[KER79] Kerr, D. S., "Data Base Machine with Large Content-Addressable Blocks and Structural Information Processes", *Computer*, Vol. 12, No. 3, March 1979, pp. 64-79.

[KHU72] Khumawala, B. M., "An Efficient Branch and Bound Algorithm for the Warehouse Location Problem", *Management Science*, Vol. 18, No. 12, Aug. 1972, pp. B718-B731.

[KLI74] Klimov, G. F., "Time Sharing Service Systems I", *Theory of Probability and its Applications*, Vol. 19, 1974, pp. 532-551.

[KNU75] Knuth, D. E., and Rao, G. S., "Activity in an Interleaved Memory", *IEEE Trans. On Comp.*, Vol. C-24, No. 9, Sept. 1975, pp. 943 - 944.

[KOO57] Koopmans, T. C. and Beckmann, M., "Assignment Problems and the Location of Economic Activities", *Econometrica*, Vol. 25, No. 1, Jan. 1957, pp. 53-76.

[KON68] Kongeim, A. G., "A Note on Time Sharing with Preferred Customers", *Z. Warsch*, Verw, Geb. 9, 1968, pp. 112-130.

[KRI78] Krishnarao, T., *A Systematic Design and Analysis of Reconfigurable Distributed Computer Systems*, Ph.D. Dissertation, University of California, Berkeley, June 1978.

[KUE63] Kuehn, A. A., and Hamburger, M. J., "A Heuristic Program for Locating Warehouses", *Management Science*, Vol. 9, No. 4, July 1963, pp.

643-666.

[LAN77] Landis, D., "Multiples-response resolution in associative systems", *IEEE Trans. Comput.*, Vol. C-26, pp. 230-235, Mar. 1977.

[LAN79] Langdon, Jr., G. G., "Data Base Machine, An Introduction", *IEEE Transactions on Computers*, Vol. C-28, No. 6, June 1979, pp. 381-383.

[LAW63] Lawler, E. L., "The Quadratic Assignment Problem", *Management Science*, Vol. 9, No. 4, July 1963, pp. 586-599.

[LEH76] Lehman, M. M., and Parr, F. N., "Program Evolution and its Impact on Software Engineering", *Proc. of the 2nd International Conference in Software Engineering*, Oct. 1976.

[LEN77] Lenstra, J. K., Rinnooy Kan, A. H. G. and Brucker, P., "Complexity of Machine Scheduling Problems", *Annals of Discrete Mathematics*, Vol. 1, North Holland Publishing Co., 1977, pp. 343-362.

[LEV74] Levin, K. D., *Organizing Distributed Data Bases in Computer Networks*, Ph.D. Dissertation, University of Pennsylvania, 1974.

[LEV75] Levin, K. D., Morgan, H. L., "Optimizing Distributed Data Bases-A Framework for Research", *Proc. NCC*, 1975, pp. 473-478.

[LEW62] Lewin, M. H., "Retrieval of ordered lists from a content-addressed memory", *RCA Rev.*, Vol. 23, pp. 215-229, June 1962.

[LIN76] Lin, C. S., et. al., "The Design of a Rotating Associative Memory for Relational Data Base Applications", *ACM Trans. on Data Base Systems*, Vol. 1, No. 1, March, 1976.

[LIP78] Lipovski, G. J., "Architectural Features of CASSM: A Context Addressed Segment Sequential Memory", *Proc. 5th Ann. Symp. on Comp. Arch.*, ACM-SIGARCH, pp. 31-38.

[LOO75] Loomis, M. E. S., *Data Base Design: Object Distribution and Resource Constrained Task Scheduling*, Ph.D. Dissertation, Comp. Sci. Dept., UCLA, 1975.

[LOO76] Loomis, M. E. S., and Popek, G. J., "A Model for Data Base Distribution", *Comp. Networks: Trends and Applications, 1976*, IEEE, pp. 162-169.

[MAH76] Mahmoud, S., Riordon, J. S., "Optimal Allocation of Resources in Distributed Information Networks", *ACM Trans. on Data Base Systems*, Vol. 1, No. 1, March 1976, pp. 66-78.

[MAN64] Manne, A. S., "Plant Location Under Economies of Scale Decentralization and Computation", *Management Science*, Vol. 11, No. 2, Nov. 1964, pp. 213-235.

[MAR75] Marill, T., and Stern, D., "The Datacomputer - A Network Data Utility", *AFIPS Conference Proceedings*, 44, 1975, pp. 389-395.

[MEI77] Meilijson, I., and Weiss, G., "Multiple Feedback at a Single Server Station", *Stochastic Processes and their Applications*, North Holland Publishing Co., Vol. 5, 1977, pp. 195-205.

[MII64] Miiller, H. S., "Resolving multiple responses in an associative memory", *IEEE Trans. Electron. Comput.* Vol. EC-13, Short Notes, pp. 614-616, Oct. 1964.

[MOE78] Moeller, A., "Fabrication Technology and Physical Fundamentals of Components used for Semiconductor Memories", *Digital Memory and Storage*, W. E. Proebster Ed., Braunschweig: Vieweg, 1978.

[MOR77] Morgan, H. L., and Levin, K. D., "Optimal Program and Data Locations in Computer Networks", *CACM*, Vol. 20, No. 5, May, 1977, pp. 315-322.

[MUN74] Muntz, R. R., et. al., "Stack Replacement Algorithms for Two Level Directly Addressable Paged Memories", *SIAM J. on Computing*, Vol. 3, No. 1, March, 1974, pp. 11-22.

[NUT77] Nutt G. J., "Memory and Bus Conflict in an Array Processor", *IEEE Trans. on Comp.*, Vol. C-26, No. 6, June 1977, pp. 514 - 521

[OZK77] Ozkarahan, E. A., et. al., "Performance Evaluation of a Relational Associative Processor", *ACM Trans. on Data Base Systems*, Vol. 2, No. 2, June 1977, pp. 175-195.

[PAR72] Parhami, B., "A Highly Parallel Computing System for Information Retrieval", *AFIPS Conf. Proc., 1972, FJCC*, Vol. 41, Part II, pp. 681-690.

[PAR73] Parhami, B., "Associative memories and processors: An overview and selected bibliography", *Proc. IEEE*, Vol. 61, pp. 722-730, June 1973.

[POH75] Pohm, A. V., "Cost/Performance Perspectives of Paging with Electronic and Electro-mechanical Backing Stores", *Proc. of the IEEE*, Vol. 63, No. 8, Aug. 1975, pp. 1123-1128.

[RAM70] Ramamoorthy, C. V., and Chandy, K. M., "Optimization of Memory Hierarchies in Multi-programmed Systems", *JACM*, Vol. 17, No. 3, July, 1970, pp. 426-445.

[RAM76] Ramamoorthy, C. V., and Krishnarao, T., "The Design Issues in Distributed Computer Systems", *Inftotech State of the Art Report on Distributed Systems, 1976*, pp. 375-400.

[RAM78a] Ramamoorthy, C. V., Turner, J. C., and Wah, B. W., "A Design of a Cellular Associative Memory for Ordered Retrieval", *IEEE Trans. on Comp.*, Vol. C-27, No. 9, Sept. 1978.

[RAM78b] Ramamoorthy C. V. and Ho, G. S., "A Design Methodology for User Oriented Computer Systems", *Proc. National Computer Conference*, AFIPS Press, 1978, pp. 953-966.

[RAM79a] Ramamoorthy, C. V., and Wah, B. W., "Data Management in Distributed Data Bases", *Proc. National Computer Conference*, AFIPS Press, 1979, pp. 667-679.

[RAM79b] Ramamoorthy, C. V., Ho, G. S., and Wah, B. W., "Distributed Computer Systems - A Design Methodology and its Applications to the Design of Distributed Data Base Systems", to appear *Infotech State of the Art Report on Distributed Systems, 1979.*

[RAM79c] Ramamoorthy, C. V., and Wah, B. W., "File Placements of Relations in a Distributed Relational Data Base", *Proc. First International Conference on Distributed Computer Systems*, Huntsville, Alabama, Oct. 1979.

[RAO77] Rao, R. C., and Rutenberg, D. P., "Multi-location Plant Sizing and Timing", *Management Science*, Vol. 23, No. 11, July 1977, pp. 1187-1198.

[RAV72] Ravi, C. V., "On the Bandwidth and Interference in Interleaved Memory Systems", *IEEE Trans. on Comp.* Vol. C-21, No. 8, Short Notes, Aug. 1972, pp. 899 - 901.

[RIT72] Ritzman, L. P., "The Efficiency of Computer Algorithms for Plant Layout", *Management Science*, Vol. 18, No. 5, Jan. 1972, Part I, pp. 240-248.

[ROS76] Ross, Sheldon M., *Introduction to Probability Models*. Academic Press, 1976.

[ROT77] Rothnie, J. B., and Goodman, N., "A Survey of Research and Development in Distributed Data Base Management" *Third Int'l Conf. on Very Large Data Bases*, 1977, pp. 48-62.

[RUD77] Rudin, H., "On Alternate Routing in Circuit Switched Data Networks", *Information Processing 77*, IFIPS, North Holland Publishing Co., 1977, pp. 321-326.

[SA 69] Sa, G., "Branch and Bound and Approximate Solutions to the Capacitated Plant Location Problem", *Operations Research*, Vol. 17, No. 6, Nov-Dec 1969, pp. 1005-1016.

[SAS75] Sastry, K. V. and Kain, R. Y., "On the Performance of Certain Multiprocessor Computer Organizations", *IEEE Trans. on Comp.* Vol. c-24. Nov. 1975, pp. 1066 - 1074.

[SAU75] Sauer, C. H. and Chandy, K. M., "Approximate Analysis of Central Server Models", *IBM J. of Research and Development*, May, 1975, pp. 301-313.

[SCH78] Schunemann, C., and Spruth, W. G., "Storage Hierarchy Technology and Organization", *Digital Memory and Storage*, W. E. Proebster ed., Braunschweig: Vieweg, 1978.

[SCH79] Schuster, S. A., et. al., "RAP.2 - An Associative Processor for Data Base and its Applications", *IEEE Trans. on Computers*, Vol. C-28, No. 6, June 1979, pp. 446-458.

[SIC77] Sickle, L. V., and Chandy, K. M., "Computational Complexity of Network Design Algorithms", *Information Processing 77*, IFIPS, North Holland Publishing Co., 1977.

[SEE62] Seeber, R. R. and Lindquist, A. B., "Associative memory with ordered retrieval", *IBM J. Res. Develop.*, Vol. 6, p. 126. Jan. 1962.

[SIL76] Siler, K. F., "A Stochastic Evaluation Model for Data Base Organization in Data Retrieval Systems", *CACM*, Vol. 19, No. 2, Feb. 1976, pp. 84-95.

[SKI69] Skinner, C. E., and Asher, J. R., "Effects of Storage Contention on System Performance", *IBM Sys. J.*, No. 4, 1969, pp. 319 - 333.

[SLA56] Slade, A. E. and McMahon, H. O., "A cryotron catalog memory system", *Proc. Eastern Joint Comput. Conf.*, Dec. 1956, pp. 115-119.

[SLO70] Slotnick, D. L., "Logic Per Track Devices", *Advances in Computers*, Academic Press, 1970, pp. 291-296.

[SMI76] Smith, A. J., *Characterizing the Storage Process and its Effects on the Update of Main Memory by Write- Through*, Research Report, University of California, Berkeley, 1976.

[SMI77] Smith, A.J., "Multi-processor Memory Organization and Memory Interference", *CACM*, Vol. 20, No. 10, Oct. 1977, pp.754-761.

[SNY71] Snyder, R. D., "A Note on the Location of Depots", *Management Science*, Vol. 18, No. 1, Sept. 1971, pp. 97.

[SPI69] Spielberg, K., "An Algorithm for the Simple Plant Location Problem with some Side Conditions", *Operations Research*, Vol. 17, Jan-Feb

1969, pp. 85-115.

[STO75] Stone, H. S., "Parallel Computers", Chapter 8, *Introduction to Computer Architecture*, H. S. Stone ed., SRA Inc., 1975.

[STO77a] Stone, H. S., "Multi-processor Scheduling with the Aid of Network Flows", *IEEE Trans. on Soft. Engr.*, Vol. SE-3, No. 1, Jan. 1977, pp. 85-93.

[STO77b] Stone, H. S., *Program Assignment in Three- Processor Systems and Tricut Partitioning on Graphs*, Report No. ECE-CS-77-7, University of Massachusetts, Amherst, Mass., 1977.

[STO78a] Stone, H. S., "Critical Load Factors in Two Processor Distributed Systems", *IEEE Trans. on Software Engineering*, Vol. SE-4, No. 3, May 1978, pp. 254-258.

[STO78b] Stone, H. S., and Bokhari, S. H., "Control of Distributed Processes", *Computer*, July 1978, pp. 97-106.

[STR70] Strecker, W. D., *Analysis of the Instruction Execution Rate in Certain Computer Structures*, Ph.D. Th., Carnegie Mellon U., Pittsburgh, Pa., 1970.

[STR77] Stritter, E., *File Migration*, Stanford Linear Accelerator Center Report, SLAC-200, Jan. 1977.

[SU 79] Su, S. Y. W., Nguyen, L. H., Emam, A., and Lipovski, G. J., "The Architectural Features and Implementation Techniques of the Multi-cell CASSM", *IEEE Trans. on Computers*, Vol. C-28, No. 6, June 1979, pp. 430-445.

[SWE76] Sweenly, D. J., and Tatham, R. L., "An Improved Long Run Model for Multiple Warehouse Location", *Management Science*, Vol. 22, No. 7, March 1976, pp. 748-758.

[TEL78] *Telenet Data Communication Network Rate Schedule*, Abstract of Telenet Tariff, FCC No. 1, Effective July 1, 1978.

[TER76] Terman, F. W., *A Study of Interleaved Memory Systems by Trace Driven Simulation*, Technical Note No. 94., Digital Systems Lab., Stanford Electronics Lab., Stanford University, Stanford, CA. 94305, Sept. 1976.

[THE78] Theis, D. J., "An Overview of Memory Technologies", *Datamation*, Jan. 1978, pp. 113-131.

[TOM67] Tomasulo, R.M., "An Efficient Algorithm for Exploiting Multiple Arithmetic Units", *IBM J. of Research and Develop.*, Jan 1967, pp.25-33.

[TUE76] Tuel, W. G., "An Analysis of Buffer Paging in Virtual Storage Systems", *IBM J. of Research and Development*, Sept. 1976, pp. 518-520.

[TUR72] Turner, J. L., *A design for a fast sorting associative memory*, Master of Science Thesis, University of Texas at Austin, Aug. 1972.

[UPT78] Upton, M., "Price/Performance Game Rules Change", *Computer World*, Jan. 23, 1978, p. 61.

[WAH76] Wah, B. W. *Analysis of Buffering in Memory Interleaving*, M. S. Report, University of Calif., Berkeley, Dec. 1976.

[WAR76] Warren, H. S. Jr., *Static Main Storage Packing Problems*, IBM Research Report, RC-6302, Nov. 1976.

[WEI63] Weinstein, H., "Proposals for ordered sequential detection of simultaneous multiple responses", *IEEE Trans. Electron. Comput.*, (Corresp.), Vol. EC-12, pp. 564-567, Oct. 1963.

[WEI77] Weide, B., "A Survey of Analysis Techniques for Discrete Algorithms", *ACM Computing Surveys*, Vol. 9, No. 4, December 1977, pp. 291-314.

[WES73] Wesolowsky, G. O., "Dynamic Facility Location", *Management Science*, Vol. 19, No. 11, July, 1973, pp. 1241-1248.

[WON76] Wong, E., and Youssefi, K., "Decomposition - A Strategy for Query Processing", *ACM Trans. on Data Base Systems*, Vol. 1, No. 3, Sept. 1976, pp. 223-241.

[WON77] Wong, E., *Restructuring Dispersed Data from SDD- 1: A System for Distributed Data Bases*, Comp. Corp. of America, Tech. Rep. CCA-77-03, 1977.

[YAN66] Yang, C. C. and Yau, S. S., "Cutpoint cellular associative memory", *IEEE Trans. Electron. Comput.*, Vol. EC-15, pp. 522-528, Aug. 1966.