INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI[®]

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Parallel I/O Scheduling on a Network of Workstations

By

Fangyu Chen, B.Eng., M.Eng.

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Engineering
in Electrical Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University

August, 2000

© 2000, Fangyu Chen, Ottawa, Canada

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-57722-8

Canada

The undersigned recommend to
the Faculty of Graduate Studies and Research
Acceptance of the thesis,

Parallel I/O Scheduling on a Network of Workstations

submitted by
Fangyu Chen, B.Eng, M.Eng.

in partial fulfillment of the requirements
for the degree of Master of Electrical Engineering

Chair, Department of Systems and Computer Engineering

Professor Shikharesh Majumdar, Thesis Supervisor

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
August, 2000

Abstract

The speed of CPU is growing continuously whereas the speeds of disk devices have not changed significantly in the past ten years. The use of multiprocessing has reduced the completion time for the “compute” component of an application. As a result the I/O subsystem is the performance bottleneck for many I/O intensive parallel applications. To improve the I/O subsystem performance and address this I/O bottleneck, parallel disk systems are used. Scheduling of parallel I/O operations has a significant impact on the performance of systems that use multiple CPU and I/O devices. The scheduling policies that work closely with the data distribution strategies and I/O model used by the application program have received little attention.

This thesis proposes new scheduling algorithms for scheduling parallel I/O in both uniprogrammed and multiprogrammed environments in which data is partially replicated. To make full use of the available I/O resources in a uniprogrammed system, two scheduling policies, Lowest Destination Degree First (LDDF) and Highest Destination Degree First (HDDF), are proposed. These policies are observed to perform very well for systems running single jobs in isolation. In a multiprogrammed parallel environment, adaptation of the algorithms to use knowledge of job characteristics is observed to produce a superior performance.

Performance prototypes are implemented on a network of workstation running under Parallel Virtual Machine (PVM). Experiments are performed with both real and synthetic applications. Insights gained into performance and scheduling with various system and workload parameters are described.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Dr. Shikharesh Majumdar for his constant guidance, inspiration, advice, and encouragement throughout this thesis project.

I would also like to thank Ms. Rosemary Daniels for her valuable suggestions on the thesis writing.

I would like to acknowledge the supports of Natural Science and Engineering Research Council (NSERC) of Canada and Communications and Information Technology Ontario (CITO).

I would also like to take this opportunity to thank my parents, my sister and my husband for their continued support, understanding, love and patience during my studies and research.

Table of Contents

CHAPTER 1 Introduction	1
1.1 Parallel Processing Technology.....	1
1.2 Motivation.....	2
1.3 Goal of the Thesis.....	4
1.4 Organization of the Thesis.....	5
CHAPTER 2 Background: Parallel I/O System.....	6
2.1 Solutions for High Performance Parallel I/O Systems.....	6
2.1.1 Hardware Architecture Improvement.....	6
2.1.2 Operating System.....	7
2.1.3 Parallel File System.....	9
2.1.4 I/O Management and Scheduling.....	9
2.2 Models for Parallel I/O.....	11
2.2.1 I/O Model.....	11
2.2.2 Data Distribution Schemas.....	13
2.2.2.1 Full Replication.....	14
2.2.2.2 No Replication.....	14
2.2.2.3 Partial Replication.....	15
2.2.2.4 Performance and Data Distribution.....	16
2.3 Workload Parameters.....	17
2.3.1 I/O Factor (F_i)	17
2.3.2 CPU Demand and I/O Demand.....	17
2.3.3 Number of Processes (n_p & n_i).....	18
2.3.4 Coefficient of Variation (Cv_p & Cv_i).....	18
2.4 System Parameters.....	19
2.4.1 Number of Devices (N_p & N_i).....	19
2.4.2 Other Definitions.....	19
2.5 Discussion.....	20

CHAPTER 3 Parallel I/O Scheduling Policies	22
3.1 I/O Routing Mechanism.....	23
3.2 Definition.....	24
3.3 Uni-programmed Environment.....	25
3.3.1 Lowest Destination Degree First (LDDF).....	26
3.3.1.1 Motivation.....	26
3.3.1.2 LDDF Algorithm.....	26
3.3.2 Highest Destination Degree First (HDDF).....	29
3.3.3 Random	30
3.4 Multi-Job Environment.....	31
3.4.1 Job-Based Scheduling Policies.....	32
3.4.1.1 Shortest Job First (SJF).....	34
3.4.1.2 Shortest Outstanding I/O Demand Job First (SOJF).....	35
3.4.2 Non-Job Based Scheduling Policies.....	36
3.4.2.1 Highest Combined Degree First (HCDF).....	38
3.4.2.2 First Come First Service Policy (FCFS).....	41
3.4.3 Example.....	42
 CHAPTER 4 Experimental Environment.....	 45
4.1 System Architecture.....	45
4.1.1 The Parallel Virtual Machine Tool.....	46
4.2 Experimental Workload	48
4.2.1 Software Architecture for Synthetic Applications.....	48
4.2.2. Pseudo-Codes of the Processes.....	51
4.2.3 Software Architecture for Real Applications.....	53
4.2.4 Implementation of Applications.....	53
4.2.4.1 Synthetic Applications.....	53
4.2.4.2 Real Applications.....	55
4.3 Types of Environment.....	56
4.3.1 Uniprogrammed Environment.....	56
4.3.2 Multiprogrammed Environment.....	56

4.4 Measurement of Performance.....	58
4.4.1 Measurement Methodology.....	58
4.4.2 Measurement Accuracy.....	58
CHAPTER 5 Performance of Parallel I/O Scheduling Strategies.....	61
5.1 Uniprogrammed Environment.....	63
5.1.1 Synthetic Application.....	63
5.1.1.1 Effects of Number of I/O Devices (N_i).....	63
5.1.1.2 Effects of Job I/O Factor F_i	67
5.1.1.3 Effects of Coefficient of Variation of I/O Demand per Request (Cv_i).....	67
5.1.1.4 Effects of Coefficient of Variation of CPU Demand per Request (Cv_p)...	69
5.1.2 Real Applications.....	71
5.1.2.1 Effects of Network Traffic.....	71
5.2 Multiprogrammed Environment.....	76
5.2.1 Synthetic Application.....	77
5.2.1.1 Effects of Multiprogramming Level M	77
5.2.1.2 Effects of Job I/O Factor F_i	78
5.2.1.3 Effects of Coefficient of Variation of I/O Demand per Request (Cv_i).....	79
5.2.1.4 Effects of Coefficient of Variation of CPU Demand per Request (Cv_p)...	80
5.2.2. Real Application.....	81
5.2.2.1 Effects of Network Traffic.....	82
5.3 Discussion.....	84
CHAPTER 6 Conclusions.....	86
6.1 Research Summary.....	86
6.2 Contributions.....	87
6.3 Future Research	88
REFERENCES.....	89
APPENDIX A.....	94
APPENDIX B.....	96

List of Figures

Figure 2.1 I/O Job Model.....	11
Figure 2.2 Full Data Replication Policy.....	15
Figure 2.3 No Data Replication Policy.....	15
Figure 3.1 Central Router for Data Partial Replication.....	24
Figure 3.2 Non-Central Router for Fixed Data Distribution.....	24
Figure 3.3 Bipartite Graph with Processor Nodes and I/O Nodes.....	25
Figure 3.4 Pseudo-Code for LDDF Scheduling Policy.....	28
Figure 3.5 Job-Based Central Router Scheduling Mechanism.....	32
Figure 3.6 Job-Based Local Scheduling Mechanism.....	32
Figure 3.7 Job-Based Combination Router Mechanism for Partial Data Replication.....	33
Figure 3.8 Non-Job Based Scheduling Mechanism.....	37
Figure 3.9 Non-Job Based Combination Router Mechanism for Partial Data Replication	37
Figure 3.10 Pseudo-Code for HCDF Scheduling Policy.....	40
Figure 3.11 An Instance of Two Jobs.....	42
Figure 3.12 Resulting Data Transfer Between Processor and I/O Nodes from Using SJF Policy	43
Figure 3.13 Resulting Data Transfer Between Processor and I/O Nodes from Using HCDF Policy.....	43
Figure 4.1 The Experimental Environment.....	46
Figure 4.2 Interaction Diagram for the Synthetic Workload.....	50
Figure 4.3 Pseudo-Code for Control Process.....	51
Figure 4.4 Pseudo-Code for I/O Process.....	52
Figure 4.5 Pseudo-Code for Compute Process	52
Figure 4.6 Software Structure Used in Multiprogrammed Environment.....	57
Figure 5.1 Performance of a Synthetic Application with $F_i=1$, $Cv_i=Cv_p=0$ on a Uniprogrammed System	64
Figure 5.2 Performance of a Synthetic Application with $F_i=2$, $Cv_i=Cv_p=0$ on a Uniprogrammed System.....	68

Figure 5.3 Performance of a Synthetic Application with $F_i=1$, $Cv_i=0.5$, $Cv_p=0$ on a Uniprogrammed System.....	70
Figure 5.4 Performance of a Synthetic Application with $F_i=1$, $Cv_i=0$, $Cv_p=0.5$ on a Uniprogrammed System.....	72
Figure 5.5 Performances of Matrix Multiplication on a Uniprogrammed System.....	74
Figure 5.6 Performance of Sorting Application on a Uniprogrammed System.....	75
Figure 5.7 Relative Performances of Scheduling Policies in a Multiprogrammed Environment with Synthetic Applications($F_i=1$, $Cv_i=Cv_p=0$, $N_i=4$).....	78
Figure 5.8 Relative Performances of Scheduling Policies in a Multiprogrammed Environment with Synthetic Applications ($F_i=2$, $Cv_p=Cv_i=0$, $N_i=4$).....	79
Figure 5.9 Relative Performances of Scheduling Policies in a Multiprogrammed Environment with Synthetic Applications ($F_i=1$, $Cv_i=0.5$ $Cv_p=0$, $N_i=4$).....	80
Figure 5.10 Relative Performances of Scheduling Policies in a Multiprogrammed Environment with Synthetic Applications ($F_i=1$, $Cv_i=0$, $Cv_p=0.5$, $N_i=4$).....	81
Figure 5.11 Relative Performances of Scheduling Policies in a Multiprogrammed Environment with Matrix Multiplication Applications ($F_i=1.1$, $N_i=4$).....	82
Figure 5.12 Relative Performances of Scheduling Policies in a Multiprogrammed System with Sorting Applications ($F_i=1.1$, $N_i=4$).....	83
Figure 5.13 Relative Performances of Scheduling Policies in a Multiprogrammed System with Sorting Applications Transferring Large Volume of Data ($M=2$)	84
Figure A.1 Performances of a Synthetic Application with $F_i=2$, $Cv_p=0.5$, $Cv_i=0$ on a Uniprogrammed System.....	94
Figure A.2 Performances of a Synthetic Application with $F_i=2$, $Cv_i=0.5$, $Cv_p=0$ on a Uniprogrammed System.....	95

Figure B.1 Relative Performances of Scheduling Policies in a Multiprogrammed Environment with Synthetic Applications ($F_i=2$, $Cv_i=0$ $Cv_p=0.5$).....	96
Figure B.2 Relative Performances of Scheduling Policies in a Multiprogrammed Environment with Synthetic Applications ($F_i=2$, $Cv_i=0.5$, $Cv_p=0$)... ..	96

List of Tables

Table 5.1 Parameters of the Experiments.....61

Chapter 1 Introduction

1.1 Parallel Processing Technology

Parallel processing, together with advances in processor speeds and memory capacities, has made the solution of many heretofore intractable problems computationally feasible. Using parallelism in the I/O subsystem is currently receiving attention. Currently, the processor technology is well ahead of I/O technology. This implies that the performance of parallel computation for many interesting classes of application is often limited by the speed of I/O rather than by the speed of computation [Patt94]. Technology and application trends indicate that this I/O bottleneck is likely to become increasingly important in the future.

Furthermore, there have been a wide variety of parallel applications that require large I/O operations [Cyph93] [Kotz94]. Such applications include scientific computing with massive data sets [Rosa94] and databases [Borl83], as well as applications which checkpoint the progress of the computations frequently, or out-of-core applications in which the main data-structure does not fit into the main memory. The relatively low speed of disk operations can severely impair the performance of the applications.

A current trend in parallel application design is to use parallel I/O with concurrent threads of execution sending concurrent disk requests to the I/O subsystem [Chou93]. Most research has focused on improving the performance of I/O devices using fairly low-level parallelism through techniques such as disk striping and interleaving. Different data layout strategies for optimizing disk performance have also been considered for multimedia applications that often impose more timing constraints on program execution in comparison to other applications. However, it seems that widely applicable solutions to the I/O bottleneck in parallel systems will require an integrated approach which addresses the problem at all levels of the system, including applications, system software, and architectures.

1.2 Motivation

In parallel system design the large discrepancy between the I/O subsystem performance and the processor subsystem performance is an important issue [Gibs91]. Explicit scheduling of parallel I/O operations is a method to reduce this performance gap, and emerges as a potentially significant contributor to an integrated approach towards solving the I/O bottleneck and improving the performance of parallel processing systems. However, scheduling parallel I/O operations has received little attention. As I/O bottlenecks become more severe, the scheduling strategy becomes increasingly important.

Most existing work on parallel I/O scheduling deals with the I/O subsystem in isolation and does not consider I/O operations in conjunction with other resource demands (e.g. for the processor and the channel) [Jain97] [Jain92]. Moreover, most of the existing work is also based on simulation results. This thesis investigates a number of integrated scheduling policies that concern scheduling of I/O requests in a single and multi-job system using measurements made on a real system instead of on a simulated system. Therefore this thesis includes the effect of system overheads that are not always possible to capture in simulation or analytic model based investigations.

Some experiments of parallel applications are run on large and expensive multi-processors, which are rare, and inaccessible to many users. As a result the interest in using distributed networks of high performance workstations, which are available to most researchers for parallel processing is growing rapidly. An investigation of how parallel I/O scheduling performs in such an environment is required. This thesis is motivated by the requirement of effective scheduling policies on such networks of workstations.

Data distribution strategies, which mean how the data is distributed across many devices, were found to have an important role in parallel I/O performance [Maju95]. Most of this work considered a full data replication. The fully replicated data system is costly in terms of disk space if the data is large. When the input and output data for applications running on an environment are very large a partial data replication strategy may be preferable to the full replication of data. Research [Diac97] shows that partial data replication provides a significantly better system performance in comparison to no data replication. This thesis focuses on how to schedule parallel I/O when data is partially replicated.

As parallel processing is becoming more popular a variety of users are becoming interested in running parallel applications. When a diverse set of programs with different workload demands are to be run the system is often multiprogrammed. By running multiple parallel applications simultaneously on the system it is possible to utilize the system resources more effectively. Current research in multiprogrammed parallel systems attests to the importance and increasing interest in multiprogramming parallel systems (see [Sevc94] for example). Thus, in addition to a single application running in isolation we have also investigated multiprogrammed environments.

1.3 Goal of the Thesis

The main objective of this thesis was to develop strategies for scheduling parallel I/Os with partial data replication in both uniprogrammed and multiprogrammed environments. These scheduling policies are based on the job characteristic to obtain a good system performance. The response time is used to provide a basis for assessment of the proposed scheduling policies [Bagr98]. The measurements were taken on a network of workstations running PVM under Linux.

Two sets of experiments were performed on a network of workstations. One set of experiments was conducted in a uniprogrammed environment; the other set was conducted in a multiprogrammed environment. Both synthetic and real applications, such as matrix multiplication and sorting applications, were used in the experiments to capture the different interprocess communication patterns in a job. In addition to various software architectures a number of workload parameters was varied in synthetic applications for

understanding the effect of these workload parameters on the performance of the scheduling policies.

1.4 Organization of the Thesis

The remainder of the thesis is organized as follows. Chapter 2 surveys existing and current research related to parallel I/Os and how to resolve the I/O bottleneck. Background information on parallel I/Os, such as I/O models and workload parameters for parallel I/O, are introduced. In chapter 3, the devising of parallel I/O scheduling policies and their algorithms under both uniprogrammed and multiprogrammed environment are presented. The environment in which the measurements are taken, and how the prototypes for the scheduling policies for application are implemented, are described in chapter 4. The results of experiments and analysis of these results are presented in chapter 5. Chapter 6 ends with some conclusions and discusses further research possibilities.

Chapter 2 Background: Parallel I/O Systems

2.1 Solutions for High Performance Parallel I/O System

Since CPU processing times have improved considerably, the performance bottleneck has been pushed towards the I/O subsystem. Current research to resolve the I/O bottleneck has focused on low-level parallelism technique such as hardware architectures and changes to operating systems.

2.1.1 Hardware Architecture Improvement

Redundant arrays of inexpensive disks (RAIDS) have been discussed in [Patt88]. Six different RAID organizations are defined. Each is characterized by differing degrees of protection against disk failure and different degrees of read/write concurrency.

In RAIDS organizations, data are interleaved across the disks, known as data striping, to provide a mechanism of recovering the data in case a disk fails. Data striping has been shown to be better than clustering data onto a single device. This is because parallelism is used and the workload is distributed across all the devices. Design and performance of RAID systems is considered in [Chen93]. When the application issues an I/O request in a

RAID architecture, the request is sent to an I/O node which has many disks. Each I/O request is served by a number of disks in parallel. This can solve the I/O bottleneck. But it should be noted that although the I/O system is parallel the application itself may not necessarily be a parallel I/O application.

2.1.2 Operating System

Research on operating systems supporting high-performance parallel I/O systems is under way [Bers98]. The operating system component offers coverage over two critical requirements for message-based massively parallel multi-computers: networking, and memory management. In networking support of multi-computers, x-kernel [Hutc91] is implemented. The x-kernel provides protocol implementors with an interface that completely hides the details of the rest of the system; this supports the rapid implementation of efficient network protocols. Memory management techniques are necessary for the current and next generations of massively parallel multi-computers. The techniques include shared virtual memory, remote memory servers, and checkpoint. Shared virtual memory, which implements coherent shared memory on a multi-computer without physically shared memory, is discussed in [Li89]. Memory Server [Ifto93] for multi-computers allows sequential programs, message-passing programs, and shared virtual memory programs to use the entire physical memory resources effectively. Checkpointing allows long running jobs to save their state from time to time so that they can be restarted in case of failures.

Research on implementing an efficient, standard interface for parallel I/O on a wide range

of computers can be found in [Jone98]. Traditional sequential interfaces for reading and writing files can create needless inefficiency by directing all data accesses to a file through a single processing node, by forcing applications to read and write data inefficiently in many small pieces, or by creating multiple independent files that must later be merged. These problems can be addressed by implementing a standard, portable programming interface called Message Passing Interface for parallel I/O (MPI-IO) and by helping developers of applications and higher-level I/O libraries use parallel I/O effectively. MPI-IO is a portability layer and a standard interface which allows the library to run on many platforms. Since vendors and others can design MPI-IO libraries that take advantage of specialized I/O features of a given parallel file system, using a portability layer can offer both applications and higher level libraries excellent I/O performance.

Overlapping memory access with computations is a standard technique for improving performance on modern architectures. A compiler based technique for overlapping access to secondary memory (disks) with computation is discussed in [Agra96]. Here, an Inter-procedural Balanced Code Placement (IBCP) framework is developed. The framework performs analysis of arbitrary recursive procedures and arbitrary control flow and replaces synchronous I/O operations with a balanced pair of asynchronous operations. It is very useful for the application that performs frequent and large accesses to secondary memory.

2.1.3 Parallel file System

Some work has been done in the area of parallel file systems. Parallel virtual file system (PVFS) used in the Dedicated Cluster Parallel Computers (DCPCs) [Ligo96] enables low cost clusters of high-performance workstations to address parallel applications with large-scale file I/O needs. PVFS is a system that provides disk striping across multiple nodes in a distributed parallel computer, and file partitioning among jobs in a parallel program. PVFS is also used to efficiently de-couple disk striping and file partitioning in a distributed parallel computing environment. Thus, it supports collective file access operations. A collective I/O request is a technique that allows processes in an application to cooperate and make a single, large request to the file system [Kotz94]. These techniques deliver high throughput over a range of configurations and have a significant impact on performance. A parallel file system is presented in [Trid98] as a means of reducing the I/O bottleneck in a supercomputer system, such as the AP 1000 multi-computer.

2.1.4 I/O Management and Scheduling

For harnessing the power of parallel I/O, research that concerns management and scheduling of parallel applications with significant I/O on a multi-programmed parallel system has been undertaken. Scheduling of programs in multiprogrammed parallel environments are discussed in [Maju94] [Maju88]. Both static and semi-static approaches to the global scheduling problem are discussed. Two I/O management issues that affect system performance in multiprogrammed parallel environments are discussed

in [Kwon96c]. One is the centralized management of I/O requests, which is appropriate for small to medium scale parallel systems, and the other is the decentralized approach, which is necessary for larger systems. The effect of application architecture and data replication strategies on performance on both uniprogrammed and multiprogrammed environments are discussed in [Diac97].

The research on parallel disk scheduling using prefetching technology is well studied [Vitt94] [Barv96]. Prefetching I/O can reduce the I/O latency seen by an application. It is especially useful in a multiple-disk I/O system because it provides a mechanism for exploiting I/O parallelism. By prefetching data from idle disks concurrently in conjunction with data demanded by the application, I/O latency of the prefetched data can be hidden. Such prefetched data is held in the I/O buffer until required by the application. A prefetching algorithm should carefully decide what data to prefetch and retain in the buffer since the amount of data that can be prefetched is limited by the size of the I/O buffer. An optimal algorithm for prefetching and I/O scheduling using a read-once model of block reference is proposed in [Kall99]. This algorithm uses a dynamic priority assignment scheme to decide when blocks should be prefetched, so as to minimize the total number of I/Os.

The existence of complex interactions among data distribution strategies, workload characteristics and scheduling is described in [kwon96a]. Based on simulation, scheduling of I/O on multiprogrammed parallel systems is analyzed; systems with and without data replication are considered. Scheduling policies which use I/O characteristics

of job are observed to perform better in comparison to policies that do not use I/O characteristics.

2.2 Models for Parallel I/O

2.2.1 I/O Model

Four I/O models are introduced in [Maju94] and [Kwon96b]. All these models have a fork and join architecture. They differ primarily in the amount of I/O parallelism and when the I/O is done. Figure 2.1 shows their structures.

Figure 2.1 I/O Job Model

(a) NO Model (b) NOP Model (c) O1 Model (d) O2 Model

No Overlap (NO) model (Figure 2.1a): A NO job begins by first reading in some data using a single read operation from a single I/O device. After the read is done, the job forks n_p children threads, each of which performs only computations. When all these

threads complete, the job performs a single write operation to a single I/O device. This write operation is to write the results of the computation. When its I/O completes, the job terminates. This job model does not have CPU-I/O overlap because the job can only do either I/O or computation, but not both simultaneously. In addition, this job does not do parallel I/O. Only a single read operation from a single job, or write operation to a single I/O device is performed.

No Overlap with Parallelism (NOP) Model (Figure 2.1b): A NOP job begins like a NO job by first reading in some data. The read, however, is done by using n_i parallel I/O operations. When all these read operations have completed, the job forks n_p children threads, each of which only performs computation. When all these threads complete their computations they join and the job performs a parallel write operation. After all the n_i write operations are completed, the job terminates. Like the NO model the NOP model has no CPU-I/O overlap. But there exists a parallel I/O operation.

Overlap-1 (O1) Model (Figure 2.1c): This job model overlaps I/O with computation. A job starts by forking n_p children threads, each of which reads its input data from an I/O device, performs some computation and then writes data to its I/O device. When all these children threads have completed they join and the job terminates. This job model has CPU-I/O overlap because the I/O performed by one thread can overlap with the computation performed by another thread at the same time. The threads are independent of each other. The O1 job model is similar to the NOP model in that both do parallel I/O. However, the difference is that there exists synchronization among children threads after finishing read or write operation in the NOP model. In an O1 job where this restriction

does not exist, a job can simultaneously do I/O and computation. Besides that, it can be observed that in a NOP job n_i can be different from n_p , whereas in the O1 model the maximum number of parallel I/O operations is constrained to be equal to n_p .

Overlap-2 (O2) Model (Figure 2.1d): This is similar to the O1 model, but it has more I/O operations per thread and has more overlap between I/O and computations. Each thread alternates between doing computations and doing I/O. The last operation a child undertakes is to write some data onto a device. When all children's last write operations finish, the threads join and the job terminates. For the O2 model there is a parameter called S , which is the number of times a thread performs I/O. Note that when $S=2$, an O2 model degenerates into an O1 model.

Many scientific programs read their input data, fork a number of processes which perform the necessary computations and then join. After that the parent process forks again and writes the results through parallel I/O operation on the disk. If such an application runs on a system which consists of a number of parallel disks it can be modeled as an NOP model.

2.2.2 Data Distribution Schemes

Data distribution has a strong impact on performance. Some work on data distribution was done by P.Kwong [Kwon96b] and L. Diaconescu [Diac98]. There are three strategies for data distribution: full replication, no replication and partial replication. Descriptions of these strategies follow.

2.2.2.1 Full Replication

In the full replication strategy, the input data read by programs is replicated on all disks. A write operation can be performed on any one disk. Any disk can be used to serve a read or write I/O request.

One way of managing I/O requests on a system with full replication is shown in Figure 2.2. All I/O requests are inserted into a central FIFO queue. A centralized scheduler monitors the status (busy or idle) of all disks in the system and sends the I/O request to the first idle I/O disk it finds. If no idle disk is available, the requests keep in the FIFO queue until a disk is idle. This centralized strategy is appropriate for small and medium scale shared memory systems or smaller distributed memory systems where the cost of maintaining a centralized database of device statuses is low. In larger systems, especially in distributed memory systems, a high cost is associated with maintaining such a centralized database and a distributed dispatching of I/O requests is more appropriate [Kwon96c]. In addition, since the parallel application deals with large amounts of data, full replication is costly in terms of disk space.

2.2.2.2 No Replication

The no replication strategy is also called fixed R/W strategy. In this strategy, there is no

Figure 2.2 Full Data Replication Policy

replication for the data read; thus there is only one disk that can serve a particular read request. Since the read request is always sent to its corresponding disk there is no need to have a centralized dispatcher or to maintain centralized database. At the same time, it reduces the disk space required (See Figure 2.3).

Figure 2.3 No Data Replication Policy

2.2.2.3 Partial Replication

Compared with the full replication strategy, with partial replication, the input data read by programs is replicated on a subset of the available disks.

2.2.2.4 Performance and Data Distribution

The experimental results provided in [Diac97], show that full replication performs the best, whereas no replication performs the worst. This is because some of the I/O operations in the application that were performed in parallel for full replication case, must be done sequentially on a system with no replication when the data accessed by the concurrent processes are located on the same disk. But the best performance using full replication strategy is at the cost of large disk space. A way to reduce the amount of disk space is the reduction of the degree of data replication. But the I/O parallelism is decreased as replication is decreased. The trade-off between data replication and performance should be considered [Diac97]. It is shown in that the performance of the partial replication strategy compares well with the performance of the full replication strategy. It indicated that it is possible to reduce the amount of data replication without incurring a large penalty in performance, as long as the degree of multiprogramming is not very large.

In this thesis, the partial replication strategy is used with both single-job and multi-job environment for NOP workload. Unless specified explicitly each input data element is assumed to be available on two disk systems. The scheduling policies are investigated on such an environment.

2.3 Workload Parameters

Workload parameters play an important role in the performance of parallel I/O application. A description of the workload parameters that affect the performance of the parallel application are given below.

2.3.1 I/O Factor (F_i)

The I/O factor is introduced in [Maju94] and is shown to have a strong impact on the performance of applications with parallel I/O. The mean ratio between the total CPU and I/O demands for all the jobs is termed the I/O factor (F_i). The total CPU demand is the sum of CPU execution times of all the processes in the application, whereas the total I/O demand is the sum of the total I/O service time experienced by each process. The definition is as following:

The I/O factor is the ratio between the mean total I/O demand and the mean total processor demand for jobs.

2.3.2 CPU Demand and I/O Demand

CPU demand is the mean CPU execution time per CPU request. I/O demand is the mean I/O execution time per I/O request.

2.3.3 Number of Processes (n_p & n_i)

A parallel I/O application is characterized by a number of threads, which perform computation only and a number of I/O operations (such as read or write). The number of CPU processes and I/O operations are represented by n_p and n_i respectively.

Clearly, if IOD represents the mean total I/O demand of a job, the mean CPU demand of a thread is cpu_d and the I/O factor is F_i , then

$$IOD = F_i * cpu_d * n_p$$

2.3.4 Coefficient of Variation (Cv_p & Cv_i)

For parallel applications there is an important factor that characterizes the variability presented in the CPU demand and I/O demand. The coefficient of variation of a CPU demand is represented by CV_p and the coefficient of variation of an I/O demand is represented by CV_i .

In this thesis the coefficient of variation of random variables is set zero or 0.5, both of them being smaller than one. So the random value generation is based on a uniform distribution. The coefficient of variation of a random variable X is defined as [Triv82]:

$$C_x = \frac{\sigma_x}{E[x]} \quad (1)$$

where σ_x is the standard deviation, $E[x]$ is the mean value

For random value X which is uniformly distributed on (a, b) , we obtain:

$$\text{Mean: } E[x] = \frac{b+a}{2} \quad (2)$$

$$\text{Variance: } \text{Var}[x] = E[x^2] - (E[x])^2 = \frac{(b-a)^2}{12} \quad (3)$$

Since $\sigma_x = \sqrt{\text{Var}[x]}$, by combining (1), (2) and (3), we get:

$$C_x = \frac{b-a}{b+a} \sqrt{\frac{1}{3}} \quad (4)$$

When $E[x]$ and C_x are given, we can use (2) and (4) to determine the values a and b , then generate the uniformly distributed random variable S_i for the service time.

$$S_i = a + (b-a) * R_i$$

where R_i is the random variable between 0 and 1.

2.4 System Parameters

2.4.1 Number of Devices (N_p & N_i)

The number of devices available to a job has an impact on performance. In this thesis, two parameters are used: the number of compute devices/nodes N_p and the number of I/O devices/nodes N_i . The impact of N_i will be investigated in this thesis that is concerned with parallel I/O.

2.4.2 Other Definitions

A set of definition for terms that will be used throughout the thesis is provided.

- a. The degree of multiprogramming (M)

The degree of multiprogramming is equal to the number of applications which are run concurrently on the system.

- b. The degree of partial data replication (K)

The degree of partial data replication is the number of disks that stored any given data element.

Note that on a real production environment the mapping of data blocks to I/O devices can be delegated to the operating system.

- c. The response time of the job

Job response time is defined as:

Response time = job finish time – job arrival time

It will include three parts: job execution time, queuing time and the inter-process communication time.

2.5 Discussion

In this survey of existing work on parallel I/O application, it can be seen that, although there is some work being done on I/O management in multi-job environment, most of the work concerning scheduling I/O is undertaken when data are fully replicated under uni-programmed or multiprogrammed environments. Most previous work is based on simulation. This thesis makes an experimental investigation of parallel I/O scheduling on a network of workstations which is composed of a set of Pentium II PC's running PVM under Linux when data is partially replicated. This thesis uses various workload

parameters to study the effect of the system parameters on the performance of scheduling policies.

Chapter 3 Parallel I/O Scheduling Policies

Scheduling parallel I/O refers to the I/O ordering of the request in the queues at I/O devices [Denn67], [Silb92].

Existing scheduling algorithms such as SSTF, SCAN and LOOK [Silb92] are used on conventional systems to select one I/O request from a list of several outstanding requests at a single I/O node. These algorithms are effective where disk seek time is significant relative to the data transfer time since they attempt to minimize the seek time associated with a disk access by reducing disk head movement.

In this thesis we consider a system where data transfer time is much greater than the access overhead. This is the case where data for a given application was stored continuously on a disk so that only a single seeks was required. Thus, the scheduling policies proposed in this thesis are no longer focused on minimizing only the access overhead since the access overhead was assumed to be relatively small. Instead the scheduling policies attempt to reduce the overall mean job response time. The scheduling policies presented in this chapter schedule parallel I/O operation for NOP workload under uniprogrammed and multiprogrammed environment. Before describing the scheduling policies background information on data replication and request routing is provided.

3.1 I/O Scheduling Mechanism

In this thesis, we discuss two kinds of data distribution. One is called partial data replication and the other is the fixed data distribution.

Using partial data replication, if the data replication degree is K , each I/O request issued from the job can be serviced by K I/O devices. In this case, a centralized scheduling mechanism was used first. This means a centralized dispatcher routes an I/O request to an I/O device by using a scheduling policy described in Section 3.3. The I/O device chosen by the centralized dispatcher would receive the I/O request. Secondly, a local dispatcher is used at each I/O node. At the I/O node, I/O requests are put into the queue maintained and accessed by the local I/O device. If the device is idle, it takes and services a request from this local FIFO queue (see Figure 3.1). If not, the request is kept in the queuing line at this I/O device.

Using fixed data distribution, only one copy of each data element is available. This means that each I/O request issued by a job is sent immediately to the queue of the I/O device for this data element. When the device becomes idle, the request at the head of the queue is removed from the local queue for servicing. An example in which data for all the processes on each CPU are available on only one of the I/O nodes is shown in Figure 3.2.

3.2 Definition

An instance of the simple I/O scheduling problem is given in Figure 3.3, which describes the data transfer between processing node (p_i) and I/O nodes (d_i). The edge T_i denotes a potential data transfer operation between a set of processors and a set of I/O devices. Note that multiple edges can exist between a processing node and I/O nodes. This is true when, due to data replication for example, the same I/O request can be satisfied by multiple nodes.

Figure 3.1 Central Scheduler for Partial Data Replication

Figure 3.2 Non-Central Scheduler for Fixed Data Distribution

Notation: Let $G = (P, D, I)$ denote a bipartite graph where P and D are two disjoint sets of vertices which correspond to the processor set and the I/O set respectively. I is the set of I/O read requests. With a partial data replication each I/O request can be served by any one of a given set of K I/O devices.

Definition 1: the degree of a vertex is the number of edges incident upon it. For example, the degree of d_1 and d_2 is three, whereas the degree of d_3 is one.

Definition 2: Each edge in the graph has a source and destination degree: the source degree is the degree of the processing node it is coming from whereas the destination degree is the degree of the I/O node it is going into. The degree of an edge is the sum of its source and destination degrees. For example, the degree of edge T_1 is five in Figure 3.3.

Figure 3.3 Bipartite Graph with Processor Nodes and I/O Nodes

3.3 Uniprogrammed Environment

A uniprogrammed environment is an environment in which there is only one job running at a time on the system. The system resources can be used exclusively by this job.

We know, when the data transfer time is much greater than the access overhead, the cost of transferring data between the processor nodes and I/O nodes has a much stronger impact on the performance of the parallel computer system. The I/O scheduling policies proposed here are intended to reduce the overall job completion time. A number of scheduling policies are provided in the following subsections. A scheduling policy can be represented by a function: $f(G) = \{i_k d_j\}$ where $\{i_k d_j\}$ is the set containing the mapping of each I/O request i_k in I to an I/O node d_j . Each scheduling policy can give rise to a different mapping. The differences in job response times produced by the scheduling policies are discussed in the later chapters of the thesis.

3.3.1 Lowest Destination Degree First (LDDF)

3.3.1.1 Motivation

When the data is partially replicated, each I/O request can be serviced by K I/O devices ($K > 1$). Due to data distribution randomness, data placement imbalance may exist on the I/O devices. If not scheduled carefully, certain I/O devices may be more frequently used by the job than other devices. This will lead to I/O bottleneck on these frequently used I/O devices. Therefore, the goal of the scheduler is to evenly spread the I/O request among K devices in order to achieve a shorter job response time. In order to achieve this, the LDDF scheduling policy tries to distribute the I/O requests to their destination I/O node in such a way that the parallelism in data transfer is maximized.

3.3.1.2 LDDF Algorithm

We use pseudo-code to describe LDDF algorithm.

Input: Bipartite graph $G=(A, B)$, where A is the set of processor nodes p_i , B is the set of I/O nodes d_i

Let D : a set of I/O nodes which receive I/O requests. $D=\{d_1, d_2, \dots, d_j\}$

P : a set of the processor nodes which sent I/O requests. $P=\{p_1, p_2, \dots, p_j\}$

D_p : a set of processor nodes which send I/O request to the I/O node d_p

$|D|$: the number of elements in set D

order(): order I/O nodes in increasing order of degree

Output: Data transfer pair between processor nodes and I/O nodes

Schedule:

begin

 get D_p for each I/O node d_p

 while requests from job processor nodes are not assigned to their I/O nodes

 begin

 initialization

 while $D \neq \{ \}$

 begin

 order (D)

 get the transfer link pair (p_i, d_i)

 update1

 update2

 end

 end

end

initialization

```
get D
get P
get degree of each node  $d_i$ 
```

get the transfer link pair (p_j, d_i)

```
get the first I/O node  $d_i$  in D
find the one corresponding node  $p_j$  in  $D_i$ 
return the data transfer link pair  $(d_i, p_j)$ 
```

update1

{ update the element in the processor set and I/O set }

```
P = P - { $p_j$ }
D = D - { $d_i$ }
```

update2

{ evict the assigned processor node and update the degree of I/O node }

```
for all  $D_p$  except for the  $D_i$  which corresponds to the assigned I/O node  $d_i$ 
begin
  if there is  $p_j$  in  $D_p$ 
     $D_p = D_p - \{p_j\}$ 
end;
decrease the degree of  $d_p$  in D by 1
delete  $d_p$  in D when the degree of  $d_p$  is 0
```

Figure 3.4 Pseudo-Code for LDDF Scheduling Policy

It can be seen that the LDDF scheduling algorithm works iteratively. In each iteration an attempt is made to map an I/O request from each processor to a different I/O node so that

these requests can be served in parallel. At the beginning of each iteration, the destination nodes are ordered in increasing order of degree. An inner loop is executed next. The I/O node, d_i , with the lowest degree is chosen first and any one of the requests for the I/O node is mapped to d_i . The next I/O node from the sorted set is then picked, in the inner loop, and an attempt is made to match it with a request coming from a processors that has not been assigned an I/O node in the current iteration (main). The algorithm stops when all requests are scheduled; another iteration is started otherwise. Note that the graph is updated after each (main) iteration and edges corresponding to assigned requests and nodes with a degree of zero are removed. An example is provided in Section 3.3.2.

3.3.2 Highest Destination Degree First (HDDF)

The HDDF scheduling policy is presented here. Instead of choosing the lowest degree I/O node first as done in LDDF, HDDF chooses the highest degree I/O node first. Therefore, the only difference in the pseudo-code is that it sorts the degree of I/O nodes in decreasing order. In each iteration, HDDF gives more importance to the I/O nodes with a higher number of requests and attempts to make sure that they are used in the current iteration.

Example: Consider the simple example in Figure 3.3 where one I/O request issued from each processor and can be served by two I/O devices. This is shown by a dark and a dotted line between one processor and two I/O nodes.

Using LDDF, we get the transfer pairs (p_3, d_3) , (p_2, d_4) , (p_1, d_1) and (p_4, d_2) in one iteration. The schedule will result in four concurrent data transfers. This is the shortest

possible schedule. A brief discussion of LDDF that produces this best schedule is presented. From the above definitions it is known that an I/O node with a smaller degree has a smaller number of processors to match with. If the LDDF policy is used, the I/O node of lowest degree (i.e. d_4) is considered first, and is certain to be used by its source processor. By doing this, it will avoid this I/O node not being used in the first data transfer round. Conversely, if this I/O node which has the lowest degree is not considered earlier and the only requests it can handle are assigned to other I/O nodes, this node will idle whereas the other more “popular” I/O nodes will have a long queue of waiting requests.

HDDF will generate the schedule pairs: (p_1, d_1) , (p_2, d_2) , (p_3, d_3) and $(p_4, d_1/d_2)$ in two iterations. An experimental comparison of the two policies for various applications characterized by different processor and I/O demands is presented in chapter 5.

3.3.3 Random Policy

Random is a low cost policy and does not use any “intelligent” scheduling mechanisms. Under the single job environment, the random policy is used as a performance yardstick for the two scheduling policies presented earlier. Random policy means that when the I/O request arrives at the central control point, the scheduler selects the destination I/O device randomly from the K possibilities and sends the request to this I/O device.

3.4 Multiprogrammed Environment

When multiple parallel jobs are submitted to the system it is possible to run them in two different ways. The jobs are said to run in a uniprogrammed fashion when they are run in sequence with the whole system dedicated to a single job until it completes. In a multiprogrammed system all the jobs start execution at the same time and share the system processing and I/O resources. It is shown in [Diac98] that the multiprogrammed system exhibits a higher system performance over the uniprogrammed systems for heterogeneous multiprogramming mixes. Thus, scheduling policies used for scheduling heterogeneous multi-I/O jobs under the multiprogrammed environment are to be devised. For a multiprogrammed environment, job-based scheduling policies, which make use of the job characteristics, can improve the system performance. This is demonstrated in [Kwon96c] through simulation results. Job-based scheduling policies use job characteristics to determine the job priority. The job with the highest priority is served by an idle I/O device immediately.

Job-based I/O scheduling can be done either centrally by a central scheduler or by a distributed scheduler. The scheduler mechanism which has been discussed in Section 3.1 works closely with the data distribution policies. When data is fully replicated, a job-based central scheduler is used. There are M I/O queues at the central scheduler, as Figure 3.5 shows. Each queue corresponds to the I/O request from one job. Once any I/O device is idle, the central scheduler chooses the first I/O request from one job queue whose priority is the highest. Then the central scheduler sends the I/O request to the I/O device.

The distributed scheduler is often used with fixed data distribution. Each local scheduler maintains its own set of M I/O queues, where M is the degree of multiprogramming. Each queue corresponds to a particular job in the multiprogramming mix. Figure 3.6 illustrates the local job-based scheduler mechanism, for I/O device is d_i . There are M queues at d_i . An I/O request from job “1” to device d_i is put in queue J_{i1} . Similarly a request from job M must be inserted in queue J_{iM} . Whenever the I/O device become idle, it will get the first I/O request from the highest priority job queue.

Figure 3.5 Job-Based Central Scheduler Scheduling Mechanism

Figure 3. 6 Job-Based Local Scheduling Mechanism

3.4.1 Job-Based Scheduling Policies

There are two kinds of job-based scheduling policies to be devised in this thesis. They are presented in this section. Since partial data replication strategy is studied here, we

propose the centralized dispatcher mechanism. A separate queue for each job is maintained at each I/O device. The arrangement is shown in Figure 3.7, assuming there are two jobs in the system.

J_1 : Job 1 queue at central scheduler J_2 : Job 2 queue at central scheduler J_{12} : Job 2 queue at local I/O node d_1

Figure 3.7 Job-Based Combination Scheduler Mechanism for Partial Data Replication

Each processor issues more than one I/O request due to more than one job running in the system at the same time. At the central scheduler, there are M job queues. The I/O requests from the same job will be put into its job queue. Since data is partially replicated, the central scheduler first uses the LDDF policy to get the I/O destination device for each I/O request of the job. By doing so, we can ensure that that if the job is executed in isolation, a small job response time can be reached. This conclusion has been discussed in the Section 3.3.2. After the processing of the central scheduler, each I/O

request can be sent to its job queue at the designated I/O device. When the local scheduler at an I/O device starts to schedule, each device will schedule the I/O request from the highest priority job queue independently. The following section will discuss how to decide the job priority at each I/O device.

3.4.1.1. Shortest Job First (SJF)

Taking advantage of inter-job CPU-I/O overlap is important on multiprogrammed environment systems in order to improve the system performance [Diac97]. The inter-job overlap is the overlap between CPU operation performed by one job with the I/O operation by another job. As will be seen later SJF improves system performance by enhancing CPU-I/O overlap for different jobs.

SJF policy assigns jobs priority based on their completion times when they run in isolation under a LDDF scheduling. Shorter the completion time of a job higher is its priority. I/O requests queued at an I/O node from a job inherits the priority of the job. Ties are broken by using a First Come First Served policy. The steps used in the SJF scheduling procedure are given as follows:

- *Step 1:* the central scheduler is used first to calculate the response time for each job under the LDDF policy, and then assign the shortest response time job the highest priority. The central scheduler uses LDDF to get the destination I/O node for each I/O request of the job, and sends it to the I/O node.

- *Step 2:* Each local scheduler at an I/O device schedules I/O request from the highest priority job queue at the I/O node first. After each local scheduler finishes scheduling the highest priority job at the I/O device the I/O request of the second highest priority job should be scheduled next and so on.

At each I/O node, the I/O request of the job that has the shortest job response time is scheduled first, which will ensure that this job can finish reading data from its I/O devices faster than other jobs for NOP model. So it is possible for this job to start to do the computation while the other jobs still undertake I/O operation. The introduced inter-job overlap between CPU and I/O will improve the average response time of the job.

3.4.1.2 Shortest Outstanding I/O Demand Job First (SOJF)

SOJF policy uses the outstanding I/O request demand of the job to determine the highest priority job. The outstanding I/O of a job demand is the sum of the demand of each request that has been issued by the job but not yet serviced. A job with a shorter outstanding I/O demand is given a higher priority over a job with a larger outstanding I/O demand. A simulation-based investigation of this policy when data is not replicated is reported in [Kwon96c]. In this policy, each local scheduler uses only the I/O requests that are waiting in its M I/O queues to get the total outstanding I/O demand for the job at I/O devices. The scheduler at each I/O device selects the first request from the job queue that has the shortest value of total outstanding I/O demand. Since total outstanding I/O demand calculated by the scheduler varies over the time, whenever the I/O device becomes idle, the job priorities must be reevaluated by the scheduler that is responsible

for this I/O device. Furthermore, each local scheduler determines job priorities using only the outstanding I/O demand information available to it, which is independent of the other schedulers at the other I/O nodes. Consequently, it is possible that the highest priority job at one I/O device may be different from that at another I/O node in SOJF policy. A description of I/O management using SOJF is provided next.

- *Step 1:* as in SJF, first the central control point uses LDDF to get the destination I/O node for each I/O request of the job, and sends the request to the destination I/O node.
- *Step 2:* whenever the I/O device is idle, each local scheduler at the I/O node selects the first I/O request from its highest priority job queue at the current time by SOJF policy.

The SOJF policy considers the job priority at each I/O device individually. This policy is analogous to the optimum processor scheduling policy, which gives the processor to the process with the smallest CPU burst. It has been proven that processing a smaller job ahead of a larger job decreases the waiting time of the smaller job more than the increased waiting time of the larger job. In the same way we expect that the I/O device selecting the shortest outstanding I/O demand job first will decrease the average job waiting time at the I/O device.

3.4.2 Non-Job Based Scheduling Policies

The non-job based scheduling policy does not assign priority to the job. At each I/O node there are no separate job queues; there is only one queue including all I/O requests from

all jobs to this node, as shown in Figure 3.8. When data is partially replicated, a centralized routing is used. An example with $M = 2$ is shown in Figure 3.9.

Figure 3.8 Non-job Based Scheduling Mechanism

Figure 3.9 Non-Job Based Combination Scheduler Mechanism for Partial Data Replication

As in the case of job-based policies, the central scheduler uses LDDF to determine the destination I/O device of each I/O request for each job. The difference between the job-based and non-job based scheduling is that for the latter policy the local scheduler maintains only one job queue and serves each request independently whereas a separate queue for each job is maintained by a job-based scheduler.

In the following sections, two non-job based scheduling policies are presented.

3.4.2.1 Highest Combined Degree First (HCDF)

Under the uniprogrammed environment, we investigated HDDF policy, which only considers the degree of the vertex at one endpoint of an edge (I/O subsystem). Now under the multiprogrammed environment, we introduce HCDF policy. This policy was first proposed and evaluated in [Jain97] where the vertices in both types of nodes in the graph in terms of data transfer were considered. This is because the “bottleneck” may arise at either the processor subsystem or at the I/O devices. The degree of the edge, which connects two vertices in the graph, is defined as the sum of the degree of the vertices at its both end. The highest priority is given to the highest degree edge.

Algorithm HCDF (from [Jain97])

Input: bipartite graph $G = (A, B, E_0)$, where A is the set of processors p_i , B is the set of I/O nodes d_j , E_0 is the set of all edges e_t which are between the source processor and the destination I/O nodes

Let: $\text{order}()$: sort the edges in descending order of degree

E_n : variable containing a set of edges which are between source processors and I/O nodes ($n \geq 0$)

D : the degree of graphs G (the degree of a graph is the maximum of the degrees of its vertices)

Output: M is the set of edges currently assigned color c

Assumption: $\text{int } c, \text{vcolor}, n$

begin

initialization

while requests from job processor nodes are not assigned to their I/O nodes

begin

$M = \{ \}$ /* M is the set of edges which are assigned color c */

$E = \text{order}(E_n)$

while E_n is not empty

begin

get the first element of edge $e_t = (p_i, d_j)$ from E_n

if $\text{vcolor}(p_i) \neq c \ \&\& \ \text{vcolor}(d_j) \neq c$

update1

else

update2

end

update3

end

end

initialization

$\text{vcolor}(p_i) = \text{vcolor}(d_j) = 0$ for all i, j ;

$E_n = \{ \}$ for all n , $1 \leq n \leq D$;

$c = 1$;

$n = 0$;

get E_0 which is the set of edges between all source processors and I/O nodes

Figure 3.10 Pseudo-Code for HCDF Scheduling Policy

The edge-coloring technique was used in [Jain96]. An edge coloring of a graph associates a color with each edge such that no two edges of the same color have a common vertex. This means that in the graph all edges colored with the same color are independent in that they have no common vertex. Hence the data transfers represented by them can be performed simultaneously. The data transfers they represent can be performed concurrently where parallel buses connect processor and I/O devices. We believe this policy is of benefit to such an architecture. An edge coloring of graph represents a schedule, where all edges with the same color represent data transfers that take place at the same time.

It has been indicated in [Jain96] that the degree of a bipartite graph is defined to be the maximum of the degree of its vertices. The minimum number of colors needed to edge-color the graph equals the degree of the graph.

It can be seen that HCDF scheduling algorithm works iteratively. Each iteration corresponds to a specific color. In each iteration it attempts to color as many edges as possible. So that these requests can be served in parallel. The number of colors required to edge-color graph determines the length of the schedule.

An example of this algorithm is presented in Section 3.4.3. When the HCDF algorithm is used for a multiprogrammed environment where data is partially replicated, LDDF is used first at the central scheduler for each job to assign the destination I/O node for each I/O request. Then HCDF is used to select the order in which the requests sent to a given I/O node will be executed. This ensures a consistency between HCDF and the job based policies.

3.4.2.2 First Come First Service Policy (FCFS)

The FCFS policy does not consider the priority of each I/O request and instead selects I/O requests in the order they are received by the local scheduler at an I/O node. Like other scheduling policies, LDDF is first used at the central scheduler to get the destination I/O node for each I/O request when data is partially replicated. Then the requests for a given I/O node are served by using FCFS at the designated I/O node.

FCFS is useful as a yardstick against which the performance of the other three policies are compared to see if any significant performance gain can be obtained by using job based I/O scheduling or not.

3.4.3 Example

Observing the four scheduling policies presented above, we notice the first step for all these policies is to use the LDDF policy at the central scheduler to get the destination I/O node for each I/O request of the job note that data is partially replicated. The reason for using LDDF first at the central scheduler is to get a shorter job response time than that achieved with HDDF or Random when this job is running in isolation from other jobs on the system.

To illustrate how these scheduling policies work, we give an example where the degree of multiprogramming is two. Since the reduction introduced by LDDF for partial data replication is the same for all these four scheduling policies, we omit the step that is implemented first in the central scheduler. Two jobs in a parallel system are shown in Figure 3.11. LDDF has been used to select the I/O node for each request emanating from a processor.

Figure 3.11 An Instance of Two Jobs

In this example job 1 has a shorter response time than that of job 2 when each job runs in isolation. This is because at destination I/O node d_3 , job 2 will have more queuing time compared with queuing time of job 1 at d_1 . Thus with the SJF scheduling policy, job 1 has the higher priority at each I/O node and the local scheduler will schedule requests from job 1 first. The result is shown in Figure 3.12.

Figure 3.12 Resulting Data Transfer Between Processor and I/O Nodes from Using SJF Policy

When the HCDF policy is used, we can get the result which is shown in Figure 3.13.

Figure 3.13 Resulting Data Transfer between Processor and I/O Nodes from Using HCDF Policy

From Figure 3.12 and Figure 3.13, we find that for NOP workload, irrespectively of SJF or HCDF, job 1 always finishes its I/O operations before job 2, which introduces inter-job overlap between CPU and I/O operation. In other words, when CPU operations are performed by job 1, the I/O operation is performed by job 2 at the same time. The introduced inter-job overlap reduces the number of simultaneous CPU or I/O requests from multiple jobs. That is, the introduced inter-job overlap between CPU and I/O

operation decreases contention for the system resources, which can improve the system performance. Furthermore, from these two figures it seems that the inter-job overlap between CPU and I/O operation introduced by SJF is greater than that by HCDF. The greater the inter-job overlap, the better the system performance is expected to be.

Chapter 4 Experimental Environment

A system which consists of multiple processors and I/O devices is used in the study. The system contains two parts: a processor subsystem and the I/O subsystem. Both real and synthetic workloads are used on the system for the evaluation of the scheduling strategies. The system and the workload are described in the following sections.

4.1 System Architecture

A cluster of workstations connected by an Ethernet network is used in the experiment. A cluster is a group of inter-connected computers working as a unified computing resource [Stal98]. A network of multiprocessor workstations is an example of a cluster system. Besides its attractive price-performance ratio, a cluster system also provides incremental scalability and high availability. So the popularity of cluster-based computing is rising rapidly. The measurement results presented in the subsequent parts of the thesis are measured on this cluster-based real network shown in Figure 4.1. The experiments use N_p compute nodes and N_i I/O nodes in this network. All the experiments were performed on a NOW consisting of a total of fifteen 233 MHz pentium PC's each running under PVM over Linux. The PC's are interconnected by a 100 Mbps Ethernet LAN. PVM is a tool which allows workstations on the network to be configured into a parallel virtual machine [Geis95]. In the experiment, each workstation in this network works as node of the

parallel virtual machine and can be configured as either an I/O node or the compute node. The communication is done by using PVM communication primitives. For I/O node, the I/O operations, which service the I/O request by reading data from or writing data to the disk, are performed using its local disk. For compute node only the CPU of the corresponding workstation is used.

Figure 4.1 The Experimental Environment

4.1.1 Parallel Virtual Machine Tool

Parallel Virtual Machine (PVM) is a software tool that permits a heterogeneous collection of computers networked together to be viewed by a user's program as a single parallel computer. It was developed at MIT [Geis95]. PVM provides a unified framework within which parallel programs can be developed in an efficient and straightforward manner using existing hardware. PVM supports a straight-forward but functionally complete message-passing model. So it can transparently handle all message routing, data conversion, and process scheduling across a network of heterogeneous systems.

The PVM system is composed of two parts. The first part is a daemon, called `pvmd`, that resides on all the computers making up the virtual machine. The second part of the system is a library of PVM interface routines. It contains a functionally complete repertoire of primitives that are needed for cooperation between processes of an application. Such user-callable routines are for message passing, spawning processes, coordinating processes, and modifying the virtual machine.

The PVM computing model is simple but very general, and accommodates a wide variety of application structures. The programming interface is deliberately straightforward, which permits simple program structures to be implemented in an intuitive manner. The user writes his application as a collection of cooperating processes. Through a library of standard interface routines, processes access PVM resources easily. These routines allow the initiation and termination of processes across the network as well as communication and synchronization between processes. Communication constructs include those for sending and receiving data structures as well as high-level primitives such as broadcast and barrier synchronization. In this thesis, the low-level data communication primitives and the broadcasting primitives are used frequently to ensure the communication and synchronization between the CPU nodes and I/O nodes.

PVM processes may possess arbitrary control and dependency structures. At any point in the execution of an application, any process in existence may start or stop other processes or add or delete computers from the virtual machine. Any process may communicate and /or synchronize with any other.

Because of its various attributes described previously and its simple and complete programming interface, PVM enables a collection of computers to be used cooperatively for performing a parallel computation. Thus it is a good tool to investigate the parallel I/O application performance in the study.

4.2 Experimental Workload

In this thesis, we consider the NOP workload, which is used in a number of scientific programs. A detailed description of the software architecture follows.

4.2.1 Software Architecture for Synthetic Applications

For the synthetic program using an NOP model we use the fork and join architecture previously described in Section 4.1.1. For the software, there are three kinds of processes: control process, compute process and I/O process. The compute processes perform the computing operations using the data obtained from the I/O processes. Each of the compute processes from a job is executed on a separate compute node. There is one I/O process on every I/O node for implementing the read or write operation on the local disk.

The network for experiment is composed of fifteen workstations. In the uni-programmed environment, six of them are used as compute nodes where the processes do their CPU operations and the other six workstations that are equipped with local disks are configured as I/O nodes. Such a system gives rise to a shared disk cluster consisting of separate compute and I/O nodes in which the maximum values of N_p and N_i are six. N_p is

fixed at six for all the experiments reported in this thesis whereas N_i is varied. The control process is executed on the thirteenth workstation and is initialized by the start of the job. The main purpose in the control process is to control the compute processes as well as I/O processes and find the destination I/O node for each I/O request using scheduling policies. The interactions among these three types of processes are explained with the help of Figure 4.2. The control process spawns a number of compute processes denoted by n_p (which is set to be six in our experiments) and a number of I/O processes denoted by n_i which is also set to be six in the experiments. After the forking procedure is finished, the control process will execute the I/O scheduling policy such as LDDF or HDDF for this application, record the starting time and signal all its compute processes to begin execution. Then the control process waits for all I/O requests from the compute processes and sends each of them to one of its destinations I/O node. In the experiment partial data replication strategy is used: each input data file is available on two I/O nodes. A write operation performed by a process can be directed to a specific I/O node whereas data can be read from any of the two nodes on which it is available. The scheduling policy implemented in the control process is used to choose one suitable I/O destination among several of them available for each I/O request. If the destination I/O node is free, the I/O request will be processed instantly. After all the read operations at each destination I/O node is completed, the control process signals all the compute processes to begin the compute operations. Again the control process waits for the completion of all its compute processes and then multicasts to all its compute processes to begin the write operations. The control process passes each write request to its destination I/O node in a way similar to the read operation. Once every compute process has finished the write

operations, the control process records the computation time, kills all the processes in this virtual machine and exits.

Figure 4.2 Interaction Diagram for the Synthetic Workload

The interaction among these three kinds of process for the NOP workload is given in Figure 4.2.

4.2.2 Pseudo Codes of the Processes

The programs written in C were developed using the parallel virtual machine. The pseudo codes for the control process, the I/O process and the compute process are presented in Figure 4.3, Figure 4.4 and Figure 4.5 respectively.

Figure 4.3 Pseudo-Code for Control Process

I/O Process

```
begin
do
begin
if a  $Cv_i$  is set to be greater then zero
begin
generate random number to compute the number of data bytes to be transferred
end
else
use a fixed number of data bytes

wait for a message from control process

set the state of I/O node to be busy
unpack the message and extract request type
if request type is a read
begin
read the desired number of data bytes from the local disk
send an acknowledge to the parent and compute process
end
else
begin
write the desired number of data bytes to the local disk
send an acknowledge to the parent and compute process
end
end
end
end
```

Figure 4.4 Pseudo-Code for I/O Process

Compute Process

```
begin
get parent id (control process id) and id of itself

wait for message of starting execution from the parent process
issue the read request command to control process
wait for start computing message from the parent
do computation

send computation done message to the parent
issue the write request command to parent

wait for acknowledge from I/O nodes
send the job done message to the parent
exit
end
```

Figure 4.5 Pseudo-Code for Compute Process

Note that in all the experiments reported in this thesis $n_p = n_i$ (see Table 5.1). Thus each compute process issues one read and one write request. For a given compute process, the same I/O device (determined by the scheduling algorithm) is used for serving both the read and write requests.

4.2.3 Software Architecture for Real Applications

The structures of the real applications are similar to the synthetic applications described in Section 4.2.1. However the real applications perform useful computation as well as different types of interprocess communications.

4.2.4 Implementation of Applications

We have used two classes of applications. One is the synthetic application class and the other is the real application class. The real application class consists of a matrix multiplication and a sorting application.

4.2.4.1 Synthetic Applications

A synthetic application uses the pseudo-code described previously in Section 4.2.2. But it does not compute any useful results. This means it only mimics the I/O operation by doing real read/write of some data from/to local disk and does dummy computations to account for the computation operation of the application. The compute process consumes CPU time just by executing dummy for loops. The CPU time consumed by the compute process is set to 20 sec. With this value, PVM overhead is just a small fraction of the

execution time and the run times for the experiments are reasonable. The I/O time is related to the number of bytes read or written. We have determined the length of data to be read or written for producing an I/O factor (F_i) equal to one. When the value of F_i is greater than one, the number of bytes read or written is scaled appropriately. For example, in the experiment when F_i is set to two, we should read/write twice the number of bytes used for F_i equal to one. But it should be noted that the compute process performs an identical amount of work both for F_i equal to one and for F_i equal to two. In synthetic applications, there is no data transfer between the compute process and I/O process. The communication between the processes is restricted only to the communication required for scheduling and synchronization, such as for request and acknowledge messages. Actual data transfers did not happen. If there is a large amount of data transferred between a compute process and an I/O process, it will introduce large network delays. In particular, because of the slow Ethernet connection between the nodes, this delay is significant for the parallel virtual machine under consideration. However, most of the commercial parallel computers use faster interconnected links and the network delay are minimal. In order to understand how the scheduling policies perform on systems with such fast inter-connect we have eliminated the data transfer delay in the synthetic applications. Moreover, the controllability of workload characteristics such as Cv_i and F_i in synthetic applications makes it easier to analyze “what if” questions.

4.2.4.2 Real Applications

Real applications consider the actual transfer of data in the network. The meaningful computation must be performed using the obtained data at the compute nodes. Thus, unlike synthetic applications, the impact of network delays is discussed in our studies. We have investigated two real applications. Each of them is briefly described.

Matrix Multiplication

This application represents a workload with high communication between compute and I/O processes. Two 2000×6000 matrices of integers are multiplied by six processes each running on a separate processor node. One of the matrices, B is to be input by every process and is replicated on all the I/O nodes. The other matrix, A is divided into six equal files and each file is replicated on two I/O nodes selected randomly from the set of N_i nodes. Note that B is replicated on all I/O devices to avoid excessive I/O contention [Yuet96]. Each process reads in its input data (B and the designated part of A) and waits until the read operations are completed by all the processes. Once this I/O operation for the entire job is complete each process computes its own result submatrix. When all the submatrices are computed the result from each process is sent to its I/O node.

Sorting

The sorting application represents a workload with lower communication between compute process and I/O in comparison to the previous application. Two types of operations are performed: sort and merge. The application sorts 12 million integers. The

data elements are divided into a number of equal sized files each of which is replicated on two I/O nodes selected randomly from the set of N_i nodes. Six processes are used for sorting. Each process running on its own processing node inputs one of the files and uses bubble sort to sort the elements it has read. At the end of the sorting phase each process sends its results to a control process. The control process is responsible for collecting the sorted components from each process and merges them to produce the result. This output data is divided into a number of equal parts and each of which is sent in parallel to the available I/O nodes. Note that the amount of data transferred between the processes is smaller compared with that for matrix multiplication.

4.3 Types of Environment

4.3.1 Uniprogrammed Environment

A uniprogrammed environment is an environment where there is only one application running at a time on the system. The system resources can be used exclusively by this application. The application structure is described in Section 4.2.1. It is possible that there is more than one I/O request arriving at one I/O node. Because these I/O requests belong to the same application, a FIFO scheduling policy is used by each I/O process.

4.3.2 Multiprogrammed Environment

A diagram showing the structure of applications in a multiprogrammed environment is given in Figure 4.6. Each job (J_i) running in the multiprogrammed system has the same software architecture described in the Section 4.2.1.

Figure 4.6 Software Structure Used in Multiprogrammed Environment

Central control running on one workstation is used to generate the multiple jobs at the same time. Considering partial data replication, we use central control to implement LDDF scheduling policy first to get the destination I/O node for each I/O request of the job. When the job-based scheduling policies such as SJF and SOJF are used there are separate queues for each job at each I/O nodes. Each I/O scheduler running at local I/O device accepts the I/O request and implements its scheduling algorithm to schedule the request. The scheduling steps are described previously in Section 3.4.1. When a non-job based scheduling policy such as HCDF is used this scheduling algorithm is implemented by central control and there is only one job queue at each I/O node. So the I/O scheduler selects I/O request in a FIFO manner as in the case of the FCFS Policy. In Figure 4.6 J_i denotes each job submitted to the system. A job includes three processes: control process, compute process and I/O process. Each job implements the same processes as described in Section 4.2.1.

4.4 Measurement of Performance

4.4.1 Measurement Methodology

The measurements reported in the thesis are made on the network of the workstations. The response time is used as the primary performance metric. The response time is the difference between the job completion time with the job arrival time. The control process uses the *gettimeofday()* system calls for the computation of response time. Note that the time taken to produce a schedule is included in the response time. The input data file is assumed to be present on the local disk at the beginning of the experiment. Since the response times of the jobs are orders of magnitude higher than the execution time for the *gettimeofday()* system call, the accuracy of measurement was not affected significantly.

The scheduling algorithms were implemented in the control process. A number of experiments were carried out to investigate the impact of various scheduling policies on performance. A factor-at-a time approach is used for conducting the experimental investigation: one parameter is varied while the others are kept constant. The results of the experiments are captured in graphs. The variable factor is plotted on the X-axis and the fixed factors are included in the caption for the figure.

4.4.2 Measurement Accuracy

The network of workstations used for measurement in this thesis is the alpha research LAN in the RADS Lab of the Systems and Computer Engineering Department, Carleton University. To avoid interference from other applications, we conducted our

measurements during the prearranged time intervals where no other users were active on the system.

Traditional measure, such as the average response time, is used to compare the performance of scheduling policies. During the experiment statistics were collected in the virtual machine. To obtain a reasonable accuracy of measurement, the job response time of an application for each input case was measured N times, and then divided by N to obtain the average response time. For workloads with $Cv_i=Cv_p=0$, N was set at 50. For workloads with a nonzero Cv_i or Cv_p , N was dependent on the confidence level and interval. In addition, each application was run for different input cases that were generated randomly. An input case refers to a particular mapping between the I/O requests and the I/O nodes for a given application. Consider an example with $K=2$. A given I/O request may be served by I/O devices 1 and 2 in a specific input case. I/O devices 3 and 4 may be eligible for serving the same request for a different input case. The final average job response time for an application is the average of the mean response time achieved with all the input cases. The number of input cases were chosen in a way to achieve a confidence interval of a $\pm 5\%$ at a confidence level of 95%. In this thesis we used a Unix shell script for controlling the experiments. The script uses various input cases and lets the application execute N times for each input cases.

Furthermore, in order to reduce the required number of measurements to obtain the desired confidence level speedily, the set of experimental results for each input case was inspected to remove all results which had highly different values from the mean. Such a difference may be attributed to the interference from periodic operating system daemons

over which we did not have any control. But the removal method only works for the value of Cv_i and Cv_p equal to zero as well as for the real applications. This is because in these cases we expect all the measurements to be comparable with one another. This method can not be applied for the Cv_i or Cv_p not equal to zero.

Chapter 5 Performance of Parallel I/O Scheduling Strategies

The performance of the parallel I/O scheduling policies are presented in this section. There are two sets of experiments: one is under the uniprogrammed environment and the other is under the multiprogrammed environment. Experiments are run using partial data replication strategy. To keep the parameter space manageable and to maintain focus on I/O, a number of parameters are held at specific values through out the experiment. A description of these parameters and their values are given in Table 5.1.

Table 5.1 Parameters of the Experiments

Parameter	Description	Values
n_i	number of I/O process	6
n_p	number of compute process	6
N_i	number of processor nodes	6
N_p	number of I/O nodes	2-6
F_i	I/O factor	1, 2
Cv_i	coefficient of variation of I/O demand	0, 0.5
Cv_p	coefficient of variation of compute demand	0, 0.5
K	degree of partial data replication	2
M	Degree of multi-programming	2-4

A number of applications were used to investigate the impact of scheduling policies on the mean job response time: synthetic, matrix multiplication and sorting. The applications were executed in both a uniprogrammed and a multiprogrammed environment. For synthetic applications, we investigated the effect of Cv_i , Cv_p and F_i . Increasing Cv_i helps to understand how the policies perform when the I/O demands of requests vary from one another. Increasing F_i is used to raise the mean total I/O demand of a job. Increasing Cv_p from zero to 0.5 is used to investigate how the policies perform when the compute demands of requests are not equal to one another. Real applications also capture the effect of data transfer on the network.

The experiments we performed are organized and presented in the following manner:

- Uniprogrammed environment
 - effects of number of I/O devices
 - effects of job I/O factor F_i
 - effects of coefficient of variation of I/O demand per request
 - effects of coefficient of variation of CPU demand per request
 - effects of network traffic
- Multiprogrammed environment
 - effects of multiprogramming level
 - effects of job I/O factor F_i
 - effects of coefficient of variation of I/O demand per request
 - effects of coefficient of variation of CPU demand per request
 - effects of network traffic

Note that in all the experiments, a partial data replication with $K=2$ was used.

5.1 Uniprogrammed Environment

A number of experiments were performed in a uniprogrammed environment in which only single job runs on the system. Results of these experiments are presented next. Note that in all the experiments on the uniprogrammed system each compute process runs on an independent node. Since this results in the same input degree for all the edges, HDDF produces the same output as HCDF.

5.1.1 Synthetic Application

5.1.1.1 Effects of Number of I/O Devices (N_i)

Figure 5.1 is the graph for $F_i=1$ and the number of I/O devices is changed from two to six.

As can be seen from the results in Figure 5.1 (a), increasing N_i causes the job response time to decrease for all the three scheduling policies. The response time decreases sharply at first but a knee is reached around $N_i = 3$ or 4. Further increase in the number of I/O nodes produces only a small improvement in performance. It demonstrates that more parallelism in I/O gives good performance.

Figure 5.1 (b) presents a comparison of the scheduling policies in terms of the ratio metric defined in Section 2.4.2. In Figure 5.1(b) it can be seen that a substantial benefit accrues from the LDDF policy, especially for N_i equal to three and six.

(a)

(b)

Figure 5.1 Performance of a Synthetic Application with $F_i=1$, $Cv_i=Cv_p=0$ in a Uniprogrammed Environment (a) Response Time (b) Ratio

Note that N_p is a multiple of N_i in these situations. For such a relationship between N_i and N_p and with partial data replication, the LDDF policy can allow the I/O request to be processed at a corresponding I/O node in a more balanced manner in comparison to other scheduling policies. (see discussion in Section 3.3.2.) This balance helps to decrease the queuing time at the I/O nodes for the I/O components of the application. For example, consider a system with N_i equal to three. With the LDDF policy the balanced distribution of I/O requests will result in an I/O device processing exactly two of the six I/O requests. For another policy, the following request distribution may results: N_1 serves 2 I/O requests, N_2 serves 3 I/O requests and N_3 serves 1 I/O request.

The completion time for I/O operations will be shorter in case of LDDF. For NOP workload and LDDF, since the I/O requests are balanced at the I/O devices, the I/O operation time is obtained by adding the two I/O requests' execution time and one request's queuing time. When other policies are used, the queuing time of I/O requests at device N_2 is larger than that at other nodes. Although N_3 only has one I/O request and there is no request waiting at N_3 , the I/O operation time is determined by I/O device N_2 . That is, the I/O time is calculated by adding three I/O requests' execution time and the two I/O request's queuing time, which is much longer than that of the LDDF.

We can get another important result from Figure 5.1. When N_i is greater than two, the ranking of scheduling policies is: LDDF, Random and HDDF. The best performance of LDDF is in agreement with the discussion in Section 3.3.2. Why is Random better than HDDF? We know that HDDF always chooses the highest degree I/O node first, then it schedules the I/O requests by using FIFO at this I/O node. Because data is partially

replicated, it is possible that the I/O request, which is being scheduled to be served at the highest degree I/O node, can also be served by another I/O node whose degree is just one. This imbalance in I/O workload distribution will result in wasting the I/O resources. Furthermore, the probability of choosing the highest degree I/O node first is larger by using HDDF than by using the Random policy. So, for a single job, Random policy is observed to perform better or an average than HDDF.

It should also be noted that when N_i is equal to two, there is almost no difference among these three scheduling policies.

The rationale is that when N_i is equal to K (K is the data partially replicated degree), the performance of the job when using these three scheduling policies is the same. In the previously discussed case, whatever scheduling policy is used, each I/O device always has three I/O requests. Furthermore, when both Cv_p and Cv_i are equal to zero, each CPU or each I/O request in the job is equal in demand. As a result the job response time is almost the same by using these three scheduling policies. For this case, we should consider the cost associated with the three scheduling policies. Note that the scheduling overheads of LDDF and HDDF are larger than that of Random. Thus Random performs slightly well than the other two scheduling policies. But it should be noted that the overheads are small for both policies when N_i is greater than two, and these overheads are no longer the main factor affecting system performance.

5.1.1.2 Effects of Job I/O Factor F_i

This experiment is concerned with the effects of I/O factor F_i on the performance of the synthetic application. The graphs obtained with $F_i = 2$ are shown in Figure 5.2.

From Figure 5.1(a) and Figure 5.2 (a), we can see that when F_i increases from one to two, the job response time increases. Clearly, when F_i increases from one to two, the I/O demand per request increases. Thus, increasing the I/O demand will cause the job response time to increase.

The important result is that the ranking of the scheduling policies is still the same: LDDF, Random, and HDDF. It can be observed from Figures 5.1 (b) and 5.2 (b) that the difference among the various scheduling policies increase when I/O demand increases. The performance benefit of LDDF policy is more pronounced at higher values of F_i .

5.1.1.3 Effects of Coefficient of Variation of I/O Demand per Request (Cv_i)

This experiment investigates the effect of coefficient of variation for the I/O request size. We set Cv_i to 0.5 by using a uniform distribution. (see Section 2.2.3.4 for reference) The results of the experiment are shown in Figure 5.3.

By comparing results in Figure 5.3 (a) with those presented in Figure 5.1 (a) we observe that when Cv_i increases from 0 to 0.5, the average job response time increases. This is because with a higher Cv_i some of the I/O requests have larger I/O demands in

(a)

(b)

Figure 5.2 Performance of a Synthetic Application with $F_i=2$, $C_{vi}=C_{vp}=0$ in a Uniprogrammed Environment (a) Response Time (b) Ratio

Comparison to their $Cv_i=0$ counterparts where each I/O request is of the same size. For NOP workload, if any read or write request has not been finished the job can not to get to its computation stage. Thus, the inequality in I/O request sizes introduced by $Cv_i=0.5$ causes a larger job response time.

It is obvious from reviewing the results shown in Figure 5.3 that LDDF still performs the best among the three scheduling policies. But sometimes HDDF behaves better than Random. A possible explanation is provided. Maximizing parallelism in data transfer is the most important factor to be considered when $Cv_i=0$. But when Cv_i is not equal to zero, each I/O request is different in size. In that case, the effect of different I/O demands will diminish the effect of maximizing I/O parallelism, which will cause HDDF to perform better than Random at some specific values of N_i . It should be noted this negative impact of variability in I/O processing is small for LDDF which still performs the best.

5.1.1.4 Effects of Coefficient of Variation of CPU Demand per Request (Cv_p)

In addition to studying the effect of coefficient of variation of I/O demand per request, the effects of coefficient of variation of CPU demand per request are also investigated. The experiment results obtained from the synthetic application are given in Figure 5.4.

The method of generating the value of 0.5 for Cv_p is the same as for $Cv_i=0.5$. Clearly, for a Cv_p of 0.5, the average job response time shown in Figure 5.4 is larger in comparison to

(a)

(b)

Figure 5.3 Performance of a Synthetic Application with $F_i=1$, $C_{vi}=0.5$, $C_{vp}=0$ on a Uniprogrammed System (a) Response Time (b) Ratio

a job with Cv_p of zero. The reason is similar to those describing the effects of coefficient of variation of I/O demand presented in the previous subsection. When Cv_p is changed from zero to 0.5, it is possible for some of CPU demands to have a larger CPU demand in comparison to their $Cv_p=0$ counterpart. For NOP workload if any compute process has not finished its computation, the job can not start its I/O operation. In our experiment, N_p was set to six and both n_i and n_p were fixed at six as well. When Cv_p is equal to zero, each of the six compute processes distributed on six different processors finished its operation at the same time, whereas a Cv_p value of 0.5 may cause at least one of these compute processes to have a longer execution time than its $Cv_p=0$ counterparts. Thus, a Cv_p value of 0.5 gives a longer job response time. The other important result is that the ranking of the scheduling policies is the same as achieved with $Cv_p = 0$.

5.1.2 Real Applications

5.1.2.1 Effects of Network Traffic

The experiment results presented previously were based on the execution of synthetic applications. The experiments with real applications, which introduce real network traffic, are presented in this section. The real applications executed are matrix multiplication and sorting running on a uniprogrammed environment. By comparing the real application results with that of the synthetic application we can further investigate the effect of message communication between the compute subsystem and I/O subsystem.

(a)

(b) Ratio

Figure 5.4 Performance of a Synthetic Application with $F_i=1$, $C_{vp}=0.5$, $C_{vi}=0$ on a Uniprogrammed System (a) Response Time (b) Ratio

The results of matrix multiplication and sorting are given in Figure 5.5 and Figure 5.6 respectively.

The average job response time is expected to increase when network traffic is introduced. The reason for this behavior is discussed here. Job response time is related to the CPU and I/O processing time, queuing time at I/O nodes and data transfer time between the compute subsystem and the I/O subsystem. For synthetic applications, only small control messages are transferred through the network. So the data transfer time is small, whereas for the real application, there is a large amount of data transferred through the network. This is the actual data needed by compute processes from/to the I/O subsystem. Thus the effects of network traffic will cause average job response times to increase.

Another important result from the experiments with real applications is that the ranking of the scheduling policies is still consistent with that obtained with the synthetic applications. But the differences in response time among these policies are small in comparison to the synthetic applications.

It should be noted that the values of I/O factor F_i reported in Figures (5.5 & 5.6) are measured on the system directly.

Results of more experiments performed with uniprogrammed systems are available in Appendix A.

(a)

(b)

Figure 5.5 Performance of Matrix Multiplication on a Uniprogrammed System

(a) Response Time (b) Ratio

(a)

(b)

Figure 5.6 Performance of Sorting Application on a Uniprogrammed System

(a) Response Time (b) Ratio

5.2 Multiprogrammed Environment

In this section we will investigate the scheduling policies used on multiprogrammed environments. Multiprogramming implies that multiple jobs are run at the same time on the system. Here, we still use the parameters given in Table 5.1 for each of the jobs in the multiprogramming mix. Each of the parallel jobs contains six child processes, which is the same as the single job described in Table 5.1. But it should be noted that although each compute process in an application is run on a separate compute node, each compute node is shared by M compute processes one from each of the jobs. The degree of multiprogramming is M which is equal to the number of jobs in the multiprogramming mix. Each I/O device may be shared by some I/O processes from the same or different jobs.

For all the experiments on the multiprogrammed environment, we were primarily concerned with the relative performances of the scheduling policies. Thus only the “ratio” metric is given in the following figures.

Under a multiprogrammed system, the degree of multi-programming M is a key factor that needs to be considered in studying the relative performances of the various scheduling policies. In the following experiments the number of I/O device N_i was set to four and the X-axis represented M . Note that $N_i = 4$ corresponds to the knee of the graph in Figure 5.1. Compared with the uniprogrammed environment the experiments needed much longer time to achieve the desired accuracy level.

5.2.1 Synthetic Application

5.2.1.1 Effects of Multiprogramming Level M

There are four kinds of scheduling policies used in the multiprogrammed environment. Two of them are job-based policies: SJF and SOJF, the other two are non-job based policies: HCDF and FCFS.

As can be seen in Figure 5.7, the ranking of the scheduling policies is SJF, SOJF, HCDF and. In other words, the job-based policies outperform the non-job based policies.

Non-Job based policies give priorities to individual requests and can lead to a larger number of unfinished jobs at a given point of time. By assigning priority on a job basis, job-based scheduling policies are tending to complete jobs quickly. We observed that for the job-based scheduling policies, SJF is better than SOJF (see Figure 5.7). The intuitive reason is that SJF gives rise to more inter-job CPU-I/O overlap. The overlap of computation performed by one job, with the I/O performed by another job, reduces the number of simultaneous I/O requests to the I/O devices. It also reduces the contention for the CPU resources from multiple jobs, and system performance improves.

Another important observation is that when M increases, the differences of the performances of the three scheduling policies with that of the FCFS strategy increase. A high gain in system performance is achieved with SJF when M is large (see Figure 5.7).

Figure 5.7 Relative Performances of Scheduling Policies in a Multiprogrammed Environment with Synthetic Applications ($F_i=1$, $Cv_i=Cv_p=0$, $N_i=4$)

Note that the average job response time increases when multiprogramming level M increases. This is expected. More contentions for CPU, I/O as well as the data link resource give rise to a larger average job response time.

5.2.1.2 Effects of Job I/O Factor F_i

As can be seen in Figure 5.8, when F_i is increased to two, the relative rankings of the policies remain consistent with that shown in Figure 5.7.

It should be noted with $F_i=2$, the difference between job-based scheduling policies and FCFS for a given M is smaller in comparison to that achieved with $F_i=1$. Similar behaviors are observed by P. Kwong [Kwon96c] based on a simulation experiment.

Figure 5.7 and Figure 5.8 show that job-based policies are better on a multiprogrammed system.

Figure 5.8 Relative Performances of Scheduling Policies in a Multiprogrammed Environment with Synthetic Applications ($F_i=2$, $Cv_i=Cv_p=0$, $N_i=4$)

5.2.1.3 Effects of Coefficient of Variation of I/O Demand per Request (Cv_i)

The rankings of SJF, SOJF, HCDF and FCFS remain unaffected as Cv_i is increased to 0.5 (see Figure 5.9).

When we compare the results seen in Figure 5.9 with those seen in Figure 5.7, we can see that the differences in performance of SJF, SOJF and HCDF with that of FCFS have

decreased when Cv_i increases to 0.5. It should be noted that the performance of SJF tends to approach that of SOJF when variability exists in I/O service times (see Figure 5.9).

Figure 5.9 Relative Performances of Scheduling Policies in a Multiprogrammed Environment with Synthetic Applications ($F_i=1$, $Cv_i=0.5$, $Cv_p=0$, $N_i=4$)

5.2.1.4 Effects of Coefficient of Variation of CPU Demand per Request (Cv_p)

Figure 5.10 shows the performance of scheduling policies at $Cv_p=0.5$.

The relative performance of the scheduling policies is the same as in the case of $Cv_p=0$. The variability in CPU demands does not have an effect on the relative performance of the scheduling policies. The only effect is that the average job response time for any given policy is higher in comparison to a workload with $Cv_p=0$. The reason is the same as the one presented in Section 5.1.1.4.

Figure 5.10 Relative Performances of Scheduling Policies in a Multiprogrammed Environment with Synthetic Applications ($F_i=1$, $C_{v_p}=0.5$, $C_{v_i}=0$, $N_i=4$)

Results of more experiments with synthetic applications are available in Appendix A. These results are consistent with the observations described in the previous sections.

5.2.2 Real Applications

We not only considered a synthetic application for the multiprogrammed system but also investigated the impact of the degree of multiprogramming on performance. The parameters used in the synthetic system are still valid, except that the I/O factor F_i was directly measured from the system.

5.2.2.1 Effects of Network Traffic

The results of experiments with matrix multiplication and sorting application are shown in Figure 5.11 and Figure 5.12 respectively.

As can be seen from Figure 5.11 and Figure 5.12, the ranking of the scheduling policies is consistent with that observed for the synthetic applications. From the above experiments, we can conclude that it is better to use job-based scheduling policies. In particular, using SJF will offer substantial gains in performance on the multiprogrammed environment.

Figure 5.11. Relative Performances of Scheduling Policies in a Multiprogrammed Environment with the Matrix Multiplication Applications ($F_i=1.1$, $N_i=4$)

We also observe that the differences among the performance of policies are higher in comparison to that obtained with synthetic applications.

No application level data transfer between CPU and I/O nodes occurred in synthetic applications where as both input and output data transfer were performed by the real applications. A higher length of messages in the network seems to produce a large difference in performance for the scheduling policies. In order to verify this conclusion, we implemented another sorting application that transferred larger data compared with that shown in Figure 5.12. Figure 5.13 illustrates the situation in which there are two jobs in the system. Comparing Figure 5.12 and Figure 5.13, if M is equal to two, we can see that when the amount of data transfer increases, the differences among the performances of the various scheduling policies increase.

Figure 5.12. Relative Performances of Scheduling Policies in a Multiprogrammed System with Sorting Application ($F_i=1.1, N_i=4$)

Note that for matrix multiplication, which transfers a large amount of data in comparison to sorting application, a higher difference among the performances of the scheduling policies is observed (see Figure 5.11).

Figure 5.13 Relative Performances of Scheduling Policies in a Multiprogrammed System with Sorting Applications Transferring Large Volume of Data (M=2)

5.3 Discussion

A summary of the major observations is provided in this section:

- LDDF policy performs best on a uniprogrammed environment.
- Job-based scheduling policies outperform non-job based scheduling policies on a multiprogrammed system.

- Introducing an inter-job CPU-I/O overlap for the NOP workload produces great benefit in a multiprogrammed environment.
- SJF performs better than SOJF policy on a multiprogrammed system, especially when large data transferring occur on the network.
- LDDF, HCDF, and HDDF are based on a priori knowledge of the software architecture of the applications. In many situations such information can be provided by the application programmer. SJF is based on a priori knowledge of job completion times which is harder to obtain. It is possible, however, to use a previously measured completion time of a job as an estimate in the current run of the job. Making such measurements are expensive and are worthwhile if the same problem is run multiple times with different input data in the same environment. Otherwise the SOJF policy that does not use any a priori knowledge of job characteristics can be used.

Chapter 6 Conclusions

6.1 Research Summary

A number of strategies for scheduling parallel I/O applications on both uniprogrammed and multiprogrammed environments were proposed and analyzed. The research was based on implementations made on a cluster of PC's running PVM under Linux. This thesis investigated the impacts of I/O scheduling policies for the class of NOP applications on performance.

A partial data replication was considered in the thesis. For a uniprogrammed environment, two scheduling policies, Lowest Destination Degree First (LDDF) and Highest Destination Degree First (HDDF), were compared. LDDF and HDDF are new policies proposed in this thesis. In a multiprogrammed environment, two job-based scheduling policies, Shortest Job First (SJF) and Shortest Outstanding Job First (SOJF) were compared with non-job based scheduling policies, Highest Combined Degree First (HCDF) and FCFS.

The performances of the scheduling policies were evaluated for a number of combinations of workload and system parameters. The effects of number of I/O devices, I/O factor, the coefficient of variation in I/O demand, the coefficient of variation in CPU

demand as well as the impact of the network traffic on system performance were investigated.

In this thesis, scheduling strategies that used a central dispatcher and local schedulers at I/O devices were analyzed. Both uniprogrammed and multiprogrammed systems with partial data replication were considered.

On a uniprogrammed environment LDDF performed better than HDDF for all values of parameters. This is because the LDDF scheduling policy can ensure that a job makes full use of its I/O resources. The superiority of LDDF was maintained when Cv_i was increased from 0 to 0.5.

On a multiprogrammed system, job-based scheduling policies performed better than non-job based policies. This is observed for systems with both synthetic and real applications. The overlap between the CPU operation of one job with the I/O operation of another job is an important factor in producing good system performance when more than one NOP jobs are submitted to the system. We found that SJF outperformed SOJF for both synthetic and real applications. The reason is that the inter-job CPU-I/O overlap introduced in SJF is greater than that achieved with SOJF.

6.2 Contributions

The main contributions made in the thesis are given in this section. Currently, little work focuses on the complex interaction between the I/O workload, the distribution of data and the I/O scheduling policies. Also, most of the previous studies of I/O scheduling policies

were based on simulation and analytic models and used data distribution strategies such as full or no replication. To the best of our knowledge no previous work exists on the performance of parallel I/O scheduling policies on a network of workstation with partial data replication that this thesis focused on.

In this thesis, two new I/O scheduling (LDDF and HDDF) policies are devised for a uniprogrammed environment in which data is partially replicated. An investigation of their performance was made on a network of workstations running PVM. It has been shown that LDDF produces a better system performance. On a multiprogrammed system, three scheduling policies were investigated. Job based scheduling policies that overlap the CPU operation of a job with I/O operation of another one proved to give rise to a high system performance. To the best of our knowledge this thesis describes the first experimental investigation of parallel I/O scheduling on a multiprogrammed system.

6.3 Future Research

In this thesis we developed scheduling policies for NOP workload where data is partially replicated. Further work can be done with other I/O models.

In the thesis, each local scheduler served I/O requests from the local I/O queue using a FIFO strategy. Instead of using FIFO, the use of a different queuing discipline such as smallest I/O request first warrants investigation. In addition, the combined I/O scheduling policy that considers both the job characteristics and the disk head location information need to be analyzed.

References

- [Agra96] G.Agrawal, A.Acharya, J. Saltz, "An Interprocedural Framework for Placement of Asynchronous I/O Operations", Proc. International Conference on Supercomputing (ICS), May 1996, pp. 358-365.
- [Bagr98] R.Bagrodia, "Input/Output: Instrumentation, Characterization, Modeling and Management Policy", Scalable I/O Initiative Working Paper No.2, 1998.
<http://www.cacr.caltech.edu/SIO/SIOpublist.html>
- [Bank96] J.Banks, J.S.Carson, B.L.Nelson, *Discrete-Event System Simulation*, 2nd edition, Prentice-Hall, New Jersey, 1996.
- [Barv96] R.D.Barve, M.Kallahalla, P.J.Varman, and J.S.Vitter, "Competitive Parallel Disk Prefectching and Buffer Management", Proc. Fifth Workshop on I/O in Parallel and Distributed Systems, San Jose, November 1997. pp. 47-56.
- [Bers98] B.Bershad, D.Black, D.Dewitt, G.Gibson et, "Operating System Support for High-Performance Parallel I/O Systems", Scalable I/O initiative Working Paper No. 4, 1998 <http://www.cacr.caltech.edu/SIO/SIOpublist.html>
- [Bora83] H.Boral, D.J. Dewitt, "Database Machines: An idea Whose Time Has Passed? A Critique of the Future of Database Machines", Proc. Third Int'l Workshop on Database Machines, 1983, pp.166-187.
- [Chen93] S.Chen, D.Townsley, "The Design and Evaluation of RAID 5 and Parity Striping Disk Array Architectures", Journal of Parallel and Distributed Computing, vol. 17, No.5, 1993, pp. 58-74.
- [Chou93] A.Choudhary, "Parallel I/O Systems", Journal of Parallel and Distributed Computing, vol. 17, No.5, 1993, pp. 1-3.

- [Cyph93] R.Cypher, P.Messina, "Architectural Requirements of Parallel Scientific Applications with Explicit Communication", Proc. International Symposium on Computer Architecture, 1993, pp. 34-45.
- [Denn67] P.J.Denning, "Effects of Scheduling on File Memory Operations," Proc. AFIPS Spring Joint Computer Conference, 1967, pp. 9-21.
- [Diac97] L.Diaconescu, "Management of Parallel I/O", M.Eng Thesis, Department of Systems and Computer Engineering, Carleton University, May 1997, Ottawa, Canada.
- [Diac98] L.Diaconescu, S.Majumdar, "The Performance of Parallel I/O on a Multiprogrammed Network of Workstations", Proc. Second IASTED International Conference on Parallel and Distributed Computing and Networks", Brisbane, December 99, pp.439-448.
- [Gibs91] G.A.Gibson, *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*, The MIT Press, Cambridge Massachusetts, 1991.
- [Hutc91] N.C.Hutchinson, L.L.Peterson, "The x-kernel: An Architecture for Implementing Network Protocols", IEEE Transactions on Software Engineering, Vol 17, No. 1, 1991, pp.64-76.
- [Ifto93] L.Iftode, K.Li, K.Peterson, "Memory Servers for Multicomputers", Proc. IEEE COMPCON Spring'93, February, 1993, pp. 538-547.
- [Jain92] R.Jain, K.Somalwar, J.Werth, J.C.Browne, "Scheduling Parallel I/O Operations in Multiple-Bus Systems," Journal of Parallel and Distributed Computing, Special issue on Scheduling and Load Balancing, Vol.3, No.12, December, 1992, pp355-365.

- [Jain97] R.Jain, K.Somalwar, J.Werth, J.C.Browne, "Heuristics for Scheduling I/O Operations", IEEE Trans. on Parallel and Distributed Systems Vol.8, No.3, March 1997, pp. 310-320.

- [Kall99] M.Kallahalla, P.J. Varman, "Optimal Read-Once Parallel Disk Scheduling", Proc. Sixth Workshop on I/O in Parallel and Distributed System (IOPADS), Atlanta, May 1999, pp.68-77.

- [Kotz94] D.Kotz, N.Nieuwejaar, "Dynamic File Access Characteristics of a Production Parallel Scientific Workload", Proc. Supercomputing '94 Conference, November, 1994, pp. 135-144.

- [Kwon96a] P.Kwong, S.Majumdar, "Scheduling of I/O in Multiprogrammed Parallel Systems", Informatica, Vol.23, No.1, April 1999, pp.67-76.

- [Kwon96b] P.Kwong, S. Majumdar, "Study of Data Distribution Strategies for Parallel I/O Management", Proc. Third International Conference of the Austrian Committee on Parallel Computing, Klagenfurt, September 1996, pp. 12-23.

- [Kwon96c] P.Kwong, "Management of I/O in Multiprogrammed Parallel Systems", M. Eng Thesis, Department of Systems and Computer Engineering, Carleton University , March 1996, Ottawa, Canada.

- [Li89] K.Li, P.Hudak, "Memory Coherence in Shared Virtual Memory Systems", ACM Transactions on Computer Systems, Vol. 7, No.4, November 1989, pp.321-359.

- [Ligo96] W.B.Ligon, R.B.Ross, "Implementation and Performance of a Parallel File System for High Performance Distributed Applications", Proc. Fifth IEEE Symposium on High Performance Distributed Computing, August 1996, pp. 178-188.

- [Maju88] S.Majumdar, D.L.Eager, R.B.Bunt, "Scheduling in Multiprogrammed Parallel Systems", Proc. 1988 ACM SIGMETRICS Conference on Measurement and modeling of Computer Systems, Santa Fe, May 1988, pp. 104-113.
- [Maju94] S.Majumdar, Y.M.Leung, "Characterization of Applications with I/O for Processor Scheduling in Multiprogrammed Parallel Systems", Proc. Sixth IEEE Symp. on Parallel and Distributed Processing, Dallas, October 1994, pp. 298-307.
- [Maju95] S.Majumdar, F.Shad, "Characterization and Management of I/O on Multiprogrammed Parallel Systems", Proc. Seventh IEEE symposium on Parallel and Distributed Processing, San Antonio, October 1995, pp. 502-510
- [Patt88] D.A.Patterson, G.Gibson, R.H.Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", ACM SIGMOD Conference, June 1988, pp. 109-116.
- [Patt94] Y.N.Patt, "The I/O Subsystem: A Candidate for Improvement", IEEE Computer, March 1994, pp. 15-16.
- [Triv82] K.Trivedi *Probability and Statistics with Reliability Queuing, and Computer Science Applications*, Prentice-Hall, NJ, 1982.
- [Geis95] A.Geist, A.Beguelin, J.Dongarra, W.Jiang, R.Manck, V.Sunderan, *PVM: Parallel Virtual Machine*, The MIT Press, Cambridge, Massachusetts, 1995.
- [Rosa94] J.M.del Rosario, A.Choudhary, "High-Performance I/O for Massively Parallel Computers: Problems and Prospects", Computer, Mar., Vol.27, No.3, 1994, pp. 59-68.
- [Sevc94] K.C.Sevcik, "Application Scheduling and Processor Allocation in Mutliprogrammed Parallel Systems", Performance Evaluation, Vol.19, No.2, March, 1994, pp. 107-140.

- [Silb92] A.Silberschatz, J.Peterson, P.Galvin, *Operating System Concepts*, 5th edition, Addison-Wesley, 1998.
- [Stal98] William Stalling, *Operating System*, 3rd edition, Prentice-Hall, 1998
- [Trid98] A.Triggell, "Parallel IO for Multicomputer Systems", 1998
available at <http://samba.anu.edu.au/~tridge/>
- [Yuet96] Yuet Ning Chan, "Processor Scheduling in a Transputer System" M.Sc. Thesis, Department of Computer Science, Carleton University, April 1996, Ottawa, Canada.
- [Vitt94] J.S.Vitter and E.A.M.Shriver, "Optimal Algorithms for Parallel Memory, I: Two-Level Memories", *Algorithmica*, Vol 12, No 2-3, February 1994, pp.110-147.

**APPENDIX A: Synthetic Applications in a Uniprogrammed System:
Additional Results**

(a)

(b)

Figure A.1 Performance of a Synthetic Application with $F_i=2$, $C_{vp}=0.5$, $C_{vi}=0$ on a Uniprogrammed System (a) Response Time (b) Ratio

(a)

(b)

Figure A.2 Performance of a Synthetic Application with $F_i=2$, $Cv_i=0.5$, $Cv_p=0$ on Uniprogrammed system (a) Response Time (b) Ratio

APPENDIX B: Synthetic Applications in a Multiprogrammed System: Additional Results

Figure B.1 Relative Performances of Scheduling Policies in a Multiprogrammed Environment with Synthetic Applications ($F_i=2$, $Cv_i=0$, $Cv_p=0.5$)

Figure B.2 Relative Performances of Scheduling Policies in a Multiprogrammed Environment with Synthetic Applications ($F_i=2$, $Cv_i=0.5$, $Cv_p=0$)