PM-1 3 1/2"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

Management of I/O in Multiprogrammed Parallel Systems

by

Peter W. T. Kwong, B. A. Sc

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

Master of Engineering

**Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada, K1S 5B6**

10 September 1996

© Copyright 1996, Peter W. T. Kwong

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395, Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

0-612-19496-5

0-612-19496-5

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

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

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

ISBN 0-612-19496-5

Canada

FIGURE C-11. R vs F_i for NOP, Round Robin allocation and higher CV_i	124
FIGURE C-12. R vs λ for NOP workloads, Replicated-C, 2 I/O devices.....	125
FIGURE C-13. R vs CV_i for NOP workloads, Replicated-C, 2 I/O devices	125
FIGURE C-14. R vs λ for OI workloads, FixedRW(RR), 4 I/O devices.....	126
FIGURE C-15. R vs CV_i for OI workloads, FixedRW(RR), 4 I/O devices	126
FIGURE C-16. R vs F_i for OI workloads, FixedRW(RR), 4 I/O devices	127
FIGURE C-17. R vs λ for NOP workloads, FixedRW(RR), 4 I/O devices.....	127
FIGURE C-18. R vs CV_i for NOP workloads, FixedRW(RR), 4 I/O devices	128
FIGURE C-19. R vs F_i for NOP workloads, FixedRW(RR), 4 I/O devices	128
FIGURE C-20. R vs λ for NOP workloads, FixedRW(RR), large n_i , 4 I/O devices.....	129

The undersigned hereby recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

Management of I/O in Multiprogrammed Parallel Systems

submitted by Peter W.T. Kwong, B. A. Sc.,
in partial fulfillment of the requirements for the degree of
Master of Engineering

Chair, Department of Systems and Computer Engineering

Professor Shikharesh Majumdar, Thesis Supervisor

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
10 September 1996

Abstract

Recent studies have concluded that I/O has become a bottleneck for a number of different classes of current and future I/O intensive parallel applications. Appropriate I/O management strategies are needed to harness the power of parallel I/O to address this bottleneck. This research focuses on some basic issues that underlie I/O management and system performance in multiprogrammed parallel environments. I/O behaviour of parallel fork-and-join applications are modelled and characterized. Based on discrete-event simulations, a number of data distribution strategies and job based I/O scheduling policies are discussed and analyzed. Results show applications which overlap computation with their parallel I/O perform better than applications without this overlap. Where feasible, centralized management for distribution of I/O requests performs better than decentralized management. Centralized data distribution strategies that use data replication are found to improve system performance in many situations. On systems in which decentralized management is required, a data distribution policy which uses no data replication demonstrates good performance. A comparison of a number of different I/O scheduling policies indicates that job based I/O scheduling can often result in lower mean job response times than policies that do not use job attributes in I/O scheduling.

Acknowledgements

About two years ago, I thought writing a thesis would be a fun experience, so I decided to work towards a thesis instead of on a project for my degree. I also decided to tackle this endeavour part-time just like the rest of my graduate courses, mainly because I already used up my allotment of full time terms going to China to study Chinese for a year. A year later, I finished my last two courses and began the long march with the research phase. Two summers later, I finally see a light at the end of the tunnel.

I wish to thank my supervisor, Professor Shikharesh Majumdar for his guidance and support during the past year and a half that this thesis was in the making. His encouragement during the periods of frustration and seeming hopelessness kept me going until my next emotional high cycle.

Many thanks to the "night shift" at the Real-Time and Distributed Systems (RADS) lab with whom I became well acquainted and whose constant source of silliness, worries and idle chitchat made coming in to school at nights bearable: Fahim, Istabrak, Mona, Tunde, Nigel, Reza, Gurudas and of course our favourite radio talk-show family therapist, Rona Raskin.

Well, one thing is for sure. The past year and a half has been quite an experience! I promise, no more simulations.....really. Tea with cardamon, anyone? Please, no more doritos.

On a more serious note, I am indebted to my family for their long distance encouragement and not disowning me even though I haven't been a model number two son. Lastly, to Bruce and Greg who have been like surrogate brothers to me. Whew! It's finally over 8-) !!

Table of Contents

Chapter 1	Introduction	1
1.1	Motivation.....	1
1.2	Thesis Objective	2
1.3	Thesis Organization	3
Chapter 2	Background: I/O Bottleneck in Parallel Computing	4
2.1	Advances in CPU and I/O Technology.....	4
2.2	I/O Intensive Applications.....	4
2.3	Hardware and Software Support for Parallel I/O.....	5
2.4	Availability of Multiprogrammed Environments.....	8
2.5	Processor Management on Multiprogrammed Systems	8
2.6	Parallel I/O Management	9
2.7	Parallel I/O Management on Multiprogrammed Systems	10
Chapter 3	The System Model and System Model Simulator.....	16
3.1	The Processor & I/O Subsystem Model Description.....	16
3.2	The Processor & I/O Subsystem Simulator Design Description	19
Chapter 4	Parallel I/O Job Models.....	27
4.1	Performance of a Single Job Running In Isolation.....	33
4.2	Performance in a Multiprogrammed Environment.....	37
4.3	Discussion	40
Chapter 5	Data Distribution Policies	42
5.1	Relative Performances of Device Allocation Policies.....	48

5.2	Relative Performances of Data Distribution Policies	52
5.3	Results for O1 workloads	52
5.4	Results for NOP workloads	60
5.5	Discussion	63
Chapter 6	Job Based I/O Scheduling Policies	66
6.1	The Performance of I/O Scheduling Policies	72
6.2	Replicated-C Results for O1 Workloads and One I/O Device.....	74
6.3	Replicated-C Results for O1 Workloads and Several I/O Devices.....	77
6.4	Replicated-C Results for NOP Workloads and One I/O Device.....	80
6.5	Replicated-C Results for NOP Workloads and Several I/O Device	86
6.6	Replicated-D Results for O1 Workloads with Several I/O Devices	89
6.7	Replicated-D Results for NOP Workloads with Several I/O Devices	94
6.8	Discussion	99
Chapter 7	Conclusions	102
7.1	Research Summary	102
7.2	Contributions	104
7.3	Suggestions for Future Research	105
	References.....	107
Appendix A	Running a Simulation	112
Appendix B	Bi-phase Hyperexponential Distribution.....	117
Appendix C	Supplemental Experimental Results.....	119

List of Figures

FIGURE 1.	High level diagram of the processor & I/O subsystem	16
FIGURE 2.	Internal structure of system simulator	20
FIGURE 3.	Data flow diagram of the simulator	23
FIGURE 4.	Fork and Join job models	28
FIGURE 5.	I/O patterns of different job models	30
FIGURE 6.	R vs number of I/O devices for all 4 job models	34
FIGURE 7.	R vs Number of NOP parallel I/O operations	36
FIGURE 8.	R vs λ for NOP, O1 and O2 job models	37
FIGURE 9.	R vs λ for increasing number of NOP I/O operations	39
FIGURE 10.	R vs λ for increasing NOP I/O operations at higher F_i	40
FIGURE 11.	Illustration of data distribution policies	44
FIGURE 12.	Device allocation policy performance with NOP workloads	49
FIGURE 13.	Round Robin vs Random device allocation for O1 workloads	50
FIGURE 14.	Random vs Round Robin device allocation for NOP workloads	51
FIGURE 15.	R vs λ for O1 workloads and Random device allocation	53
FIGURE 16.	R vs λ for O1 workloads and Round Robin device allocation	54
FIGURE 17.	R vs CV_i for O1 workloads and Random device allocation	55
FIGURE 18.	R vs CV_i for O1 workloads and Round Robin device allocation	56
FIGURE 19.	R vs F_i of O1 workloads with Random device allocation	57

FIGURE 20.	R vs F_i for OI workloads with Random allocation at higher CV_i .	58
FIGURE 21.	R vs F_i for OI workloads and Round Robin device allocation	59
FIGURE 22.	R vs F_i for OI workloads, Round Robin allocation and higher CV_i .	59
FIGURE 23.	R vs λ for NOP workloads and Random device allocation	60
FIGURE 24.	R vs λ for NOP workloads and Round Robin device allocation	61
FIGURE 25.	No job based scheduling versus job based scheduling	67
FIGURE 26.	R vs λ for OI workload, Replicated-C, 1 I/O device	74
FIGURE 27.	R vs CV_i for OI workloads, Replicated-C, 1 I/O device	76
FIGURE 28.	R vs F_i for OI workloads, Replicated-C, 1 I/O device	78
FIGURE 29.	R vs λ for OI workloads, Replicated-C, 4 I/O devices	79
FIGURE 30.	R vs CV_i for OI workloads, Replicated-C, 4 I/O devices	80
FIGURE 31.	R vs F_i for OI workloads, Replicated-C, 4 I/O devices	81
FIGURE 32.	R vs F_i for OI workloads, Replicated-C, 4 I/O devices at higher CV_i .	82
FIGURE 33.	R vs λ for NOP workloads, Replicated-C, 1 I/O devices	83
FIGURE 34.	R vs CV_i for NOP workload, Replicated-C, 1 I/O device	86
FIGURE 35.	R vs F_i for NOP workloads, Replicated-C, 1 I/O Device, higher CV_i .	87
FIGURE 36.	R vs λ for NOP workloads, Replicated-C, 4 I/O devices	88
FIGURE 37.	R vs CV_i for NOP workloads, Replicated-C, 4 I/O devices	89
FIGURE 38.	R vs F_i for NOP workloads, Replicated-C, 4 I/O devices, low CV_i .	90
FIGURE 39.	R vs F_i for NOP workloads, Replicated-C, 4 I/O devices, higher CV_i .	90
FIGURE 40.	R vs λ for OI workloads, Replicated-D and 4 I/O devices	91

FIGURE 41.	R vs CV_i for OI workloads, Replicated-D and 4 I/O devices	92
FIGURE 42.	R vs F_i for OI workloads, Replicated-D and 4 I/O devices	93
FIGURE 43.	R vs λ for NOP workloads, Replicated-D and 4 I/O devices.....	94
FIGURE 44.	R vs CV_i for NOP workloads, Replicated-D and 4 I/O devices	96
FIGURE 45.	R vs F_i for NOP workloads, Replicated-D, low CV_i , 4 I/O devices.....	97
FIGURE 46.	R vs F_i for NOP workloads, Replicated-D, higher CV_i , 4 I/O devices.....	98
FIGURE 47.	R vs λ for NOP workload, Replicated-D, large n_i , 4 I/O devices	99
FIGURE 48.	R vs CV_i for NOP workloads, Replicated-D, large n_i , 4 I/O devices	100
FIGURE A-1.	Flow diagram of a batch simulation run	112
FIGURE A-2.	Sample input parameter data file 'test.dat'	115
FIGURE A-3.	Sample batch variables file 'bat.vars'	116
FIGURE C-1.	R vs Number of I/O devices for all 4 job models at low F_i ,	119
FIGURE C-2.	R vs Number of NOP parallel I/O operations at low F_i ,	120
FIGURE C-3.	Performance of job models at higher F_i ,	120
FIGURE C-4.	Device allocation policy performance for OI workloads	121
FIGURE C-5.	R vs λ for NOP workloads, Random allocation, large $n_i=20$	121
FIGURE C-6.	R vs CV_i for NOP workloads and Random device allocation	122
FIGURE C-7.	R vs CV_i for NOP workloads and Round Robin device allocation	122
FIGURE C-8.	R vs F_i for NOP workloads, Random allocation, low CV_i ,	123
FIGURE C-9.	R vs F_i for NOP workloads, Round Robin allocation, low CV_i ,	123
FIGURE C-10.	R vs F_i for NOP workloads, Random allocation and higher CV_i ,	124

FIGURE C-11. R vs F_i for NOP, Round Robin allocation and higher CV_i	124
FIGURE C-12. R vs λ for NOP workloads, Replicated-C, 2 I/O devices.....	125
FIGURE C-13. R vs CV_i for NOP workloads, Replicated-C, 2 I/O devices	125
FIGURE C-14. R vs λ for OI workloads, FixedRW(RR), 4 I/O devices.....	126
FIGURE C-15. R vs CV_i for OI workloads, FixedRW(RR), 4 I/O devices	126
FIGURE C-16. R vs F_i for OI workloads, FixedRW(RR), 4 I/O devices	127
FIGURE C-17. R vs λ for NOP workloads, FixedRW(RR), 4 I/O devices.....	127
FIGURE C-18. R vs CV_i for NOP workloads, FixedRW(RR), 4 I/O devices	128
FIGURE C-19. R vs F_i for NOP workloads, FixedRW(RR), 4 I/O devices	128
FIGURE C-20. R vs λ for NOP workloads, FixedRW(RR), large n_i , 4 I/O devices.....	129

List of Tables

TABLE 1.	Parameters characterizing the system model	16
TABLE 2.	Common set of workload model parameters	32
TABLE 3.	Mean I/O and CPU demands per request for different job models.....	33
TABLE 4.	Example of applying job based I/O scheduling	72

List of Terms and Definitions

- FixedR-C:** A centralized fixed-read-write-anywhere data distribution policy. Reads can be serviced only by a specific I/O device but writes can be serviced by any device. Write requests are managed by a centralized router which maintains a database of I/O device statuses. Write requests wait in a central queue until a device is idle.
- FixedR-D:** A distributed fixed-read-write-anywhere data distribution policy. Reads can be serviced only by a specific I/O device but writes can be serviced by any device. Write requests are sent to a device chosen at random without knowing if the device is idle or busy.
- FixedRW:** A fixed-read-fixed-write data distribution policy. Reads and writes can be serviced only by a specific I/O device. Requests are sent to the device without knowing if the device is idle or busy.
- FixedRW(Rand):** A FixedRW policy. Both read and write requests are sent to specific I/O devices. In the simulation model, data/devices are allocated in a random manner to a job.
- FixedRW(RR):** A FixedRW policy. Both read and write requests are sent to specific I/O devices. In the simulation model, data/devices are allocated in a round robin manner.
- JBFIFO:** A job based I/O scheduling policy where priority is given to the job with the earlier job arrival time into the system.
- JBLOIO:** A job based I/O scheduling policy where priority is given to the job with the larger total outstanding I/O demand.

- JBNOT:** An I/O scheduling policy that does not use information regarding the state of a job. Individual I/O requests from a job are serviced in a First Come First Served basis rather than based on job priority.
- JBRR:** An I/O scheduling policy where each job is serviced in a round robin manner.
- JBSOIO:** A job based I/O scheduling policy where priority is given to the job with the smaller total outstanding I/O demand.
- NO:** A fork and join model of a parallel application which does I/O. This model does not do parallel I/O. Read I/O is completed before the job creates the children, and write I/O is started after all children threads complete their computations.
- NOP:** A fork and join model of a parallel application which does I/O. This model does parallel I/O, but does not overlap I/O with computation. Read I/O is completed before the job creates the children, and write I/O is started after all children threads complete their computations.
- O1:** A fork and join model of a parallel application which does I/O. This model does parallel I/O and can overlap I/O with computation. All I/O and computation are done by children threads created by the job. A child thread only does one read at the beginning of execution and one write I/O operation at the end of execution.
- O2:** A fork and join model of a parallel application which does I/O. This model is like the O1 model but each child thread can do several read or write operations.
- Random Device Allocation:** A device allocation policy where devices are chosen at random and assigned to a job. Alternatively, it refers to the placement of data on randomly selected I/O devices.

- Replicated-C:** A centralized write-anywhere-read-anywhere data distribution policy. Reads and writes can be serviced by any I/O device. I/O requests are managed by a central router which maintains a database of I/O device statuses. Requests wait in a central queue until a device is idle.
- Replicated-D:** A distributed write-anywhere-read-anywhere data distribution policy. Reads and writes can be serviced by any I/O device. I/O requests are sent to a device chosen at random without knowing whether the device is idle or busy.
- Round Robin Device Allocation:** A device allocation policy where devices are chosen sequentially from a list of available devices and assigned to jobs. Alternatively, it refers to the distribution of data across all available I/O devices to improve I/O parallelism.

Chapter 1 Introduction

1.1 Motivation

Advances in computing technology have dramatically improved the performance of hardware components such as the central processor unit (CPU) and memory but improvements in I/O device technology have not been as dramatic. Multiprocessor systems to run parallel applications already exist today and general multiprocessor systems which can run a variety of applications concurrently are becoming popular. Such a system which can have multiple different applications active at the same time can appeal to different user communities and enhance system resource utilization. The effectiveness of multiprocessor systems is still limited by the I/O subsystem because of the slow improvement in I/O technology, particularly for the electro-mechanical disk devices which are widely used today. Strategies are needed to address and narrow this disparity between processing and I/O performance.

Much research has been done on processor resource management covering areas such as local and global processor scheduling, dynamic and static allocation of processors and task decomposition into parallel structures to take advantage of multiprocessor systems, all with the focus on improving job throughput and response time. Despite the performance disparity between processing and I/O, relatively little research has been done on managing I/O to achieve the same aforementioned goals of improving job throughput and response time. A set of effective I/O management strategies that can be incorporated into the operating system of a multiprocessor system is needed to manage the I/O bottleneck

situation so as to harness more of the potential power of parallel computing systems. High performance applications which require gigabytes of data and which require data throughputs of many megabytes per seconds already exist today[Delr94]. Applications with even greater demands on the I/O system are predicted for the future. The research for this thesis is motivated by such a need to effectively manage parallel I/O to handle the I/O bottleneck of current and future applications.

1.2 Thesis Objective

The exploratory research for this thesis takes an integrated approach to parallel I/O management in a multiprogrammed environment. The objective is to find methods to address the I/O bottleneck issue to improve system performance. A new I/O application model and various types of new I/O management policies are proposed.

The research considers three main issues: the workload applied to the system that captures different ways in which applications perform I/O; the use of data distribution strategies; the use of job based I/O scheduling to manage I/O requests entering the I/O subsystem. Each issue is studied within the context of the other two to gain insights into the system behaviour from which general trends in I/O management may be derived.

The workload models' relative performances and behaviour are first characterized in a multiprocessor, multiprogrammed system. Then different workloads are applied to the system to investigate the impacts of various data distribution policies on system performance. Finally, different workloads and data distribution policies are combined with the

various proposed job based I/O scheduling policies to investigate their combined effects on system performance.

1.3 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 reviews the literature on research related to parallel I/O and the I/O bottleneck problem and describes in detail the research problem of this thesis in relation to the current literature. Chapter 3 describes the model of the multiprocessor, multiprogrammed system used as the basis for this investigation. A high level design description of the computer simulation program used to simulate the system model, the job models, the distribution policies and the scheduling policies is also given. The different fork and join I/O models are introduced and a study of their performance characteristics is given in Chapter 4. Study of I/O management begins with a study of the device allocation and data distribution policies in Chapter 5, followed by a study of the job based I/O scheduling policies in Chapter 6. Finally, findings from this research and suggested areas for further research are presented in Chapter 7.

Chapter 2 Background: I/O Bottleneck in Parallel Computing

2.1 Advances in CPU and I/O Technology

Processing power as well as memory speed for computing systems have been increasing continuously. In addition to the increase in CPU speed, the past decade has experienced a large improvement in computation speedup achieved through parallelizing applications and running these on a multiprocessor system. Consequently, the CPU processing times of computationally intensive programs have improved considerably.

The improvements in hard disk technology mainly have achieved increases in storage density and size. Because of their electro-mechanical construction, the access time for disks has improved only minimally over the past twenty years [Patt94]. This slow improvement in I/O device performance has not kept pace with that of CPU and memory, thereby pushing the performance bottleneck towards the I/O subsystem.

2.2 I/O Intensive Applications

Studies show that significant I/O is done by a variety of classes of parallel applications that include graphics software [Cyph93] [Kotz 94], the Grand Challenge problems (such as computational fluid and combustion dynamics) and scientific applications with I/O throughput requirements in the order of hundreds of megabytes per second (such as climate prediction) [Delr94]. Multimedia applications such as Media-On-Demand also have heavy I/O demands where I/O throughput rates in the order of 0.5 to 80 Mbps are required [Gemm95]. Multimedia applications also have the added complication that they must be able to handle several concurrent request streams plus satisfy media synchronization and

real-time scheduling constraints to maintain a specific quality of service [Gemm95], [Redd94].

2.3 Hardware and Software Support for Parallel I/O

Current research to increase I/O throughput has focused on hardware architectures and changes to operating systems to support parallel I/O. Redundant arrays of inexpensive disks (RAIDs) have been proposed to address the impending I/O crisis [Patt88] [Katz89]. Six different RAID organizations are defined, each with differing degrees of protection against disk failures and each with different degrees of read/write concurrency. Data are interleaved across the disks together with any parity information needed by a particular RAID organization to recover the data in case a disk fails. This interleaving of data across several devices is known as data striping (when the granularity is small) or data declustering (when the granularity is large) and has been shown to be better than clustering data onto a single device because of the use of parallelism and the more even distribution of workload across all the devices [Kim86] [Livn87] [Sale86].

Level 1, 2 and 3 RAID configurations are suitable for large data transfers because all the disks in a group are involved in each I/O operation. Level 4 RAID improves performance for small data transfers by storing whole data transfer units onto a single sector rather than bit-interleaving the transfer unit across all disks. This allows the disk array to do more than one I/O operation at a time although only one write operation can be performed at a time because one check disk is still used by RAID 4. Level 5 RAID eliminates this limitation by distributing the data and check information across all the disks. This

configuration allows a group of disks to handle multiple writes and reads at the same time and is suitable for both large data transfers (supercomputing applications) or small data transfers (transaction processing).

Research covering various design issues relating to level 5 RAID have been done by various researchers. [Chen93] investigates the performance difference between a RAID 5 and parity striped disk array and proposes two solutions for the write synchronization problem associated with RAID 5 architectures. [Chen95] investigates scheduling policies for RAID 1 type arrays and solutions for write synchronization for RAID 5 arrays. Implications of different parity distribution schemes for RAID 5 are studied in [Lee93].

RAIDs are an integral part in current designs of multimedia file servers proposed by [Jada95] [Loug93] [Ghan93] as a means to increase I/O parallelism to reduce the I/O bottleneck during retrieval of multimedia data. [Jada95] uses round robin data striping of data to increase parallelism while [Loug93] uses data striping and disk layout to achieve low seek overhead and efficient disk space allocation. [Ghan93] investigates different ways of replicating multimedia data on the disk arrays to increase I/O parallelism.

A number of different I/O subsystem architectures that are of importance in the context of shared memory and distributed memory systems are discussed by [Delr94]. [Feit95] examines architectures and implementation issues of parallel I/O subsystems suitable for massively parallel computers. Architectures include attaching disks directly to compute nodes (ie. tightly coupled), attaching disks and compute nodes to an interconnection network so all disks are accessible by all compute nodes (ie. loosely coupled), and intercon-

necting compute nodes with I/O nodes which control the disks directly, of which the Intel Touchstone Delta is an example [Delr94].

Because data retrieved from an I/O subsystem typically passes through an operating system before reaching an application, excessive physical copying of data can cause performance bottlenecks in a computing system. [Kric94] and [Pasq94b] report changes made to an operating system to address this issue of physical data movement that resulted in I/O intensive applications running faster. Other aspects of the operating system such as real-time scheduling and resource management which are critical to a multimedia system have been surveyed by [Ste95].

The design of an operating system's file systems is a key factor in the overall effectiveness of a parallel I/O subsystem [Delr94]. Parallel file systems should be able to take advantage of the underlying parallel I/O devices, but also be able to map the data decomposed by an application in the problem domain into the device storage domain. A two-phase data access strategy in which the data distribution on computational nodes is decoupled from storage distribution is proposed in [Delr93]. File partitioning for an inverted file system distributed across multiple disks in a multiprocessor share-everything environment is presented by [Jeon95]. [Corb93] introduces the Vesta File System which addresses the I/O problems of massively parallel computers running high performance scientific applications requiring concurrent access of large files by parallel processes.

2.4 Availability of Multiprogrammed Environments

A large number of commercial shared memory as well as distributed memory multiple processor systems are currently in use. Examples include shared memory systems produced by Sequent and Encore in which processes or threads in an application exchange information through shared variables stored in the global memory provided on such a system. Distributed memory systems on which processes communicate through message passing include the Intel's Hypercube and the NCube systems. A hybrid between these two classes is the class of non-uniform-memory-access (NUMA) systems such as the Tera-computer, and the KSR-1. The availability of parallel hardware along with tools such as restructuring compilers for developing parallel application software are increasing the popularity of parallel systems. Many multiprocessor systems are dedicated to running a single computationally intensive application in isolation. The usage of multiprocessor systems, however, is becoming more and more widespread so general purpose systems that run a variety of applications are becoming popular. Environments running such a variety of different applications need to *multiprogram* the system such that multiple parallel applications are active simultaneously in order to satisfy different users' computational needs and enhance resource utilization.

2.5 Processor Management on Multiprogrammed Systems

Work on resource management on multiprogrammed parallel systems has concentrated on how to schedule the processing resources among competing applications to achieve good system performance such as small mean job response time. Static scheduling strat-

egy has been investigated by a number of researchers (see [Sevc94], [Maju91] for example) and is effective in environments which have a large cost associated with switching processors among different applications. Examples of such environments are large scale NUMA shared memory systems and distributed memory systems. Processors allocated to the same job are shareable by the threads in the job that share the same address space.

Research on scheduling in shared memory systems has addressed issues such as the performance of a number of dynamic scheduling policies [Maju88] [Leut90], and characterization of parallel applications for scheduling [Ghos90] [Maju91] [Sevc94]. Examples of research in scheduling on distributed memory systems include the dynamic partitioning of processing resources in a Transputer environment [Duss90], and the advantages of time sharing processors among a number of applications [Seti93]. Scheduling of memory constrained jobs are considered in [McCa95] whereas the interaction between memory management and processor scheduling is captured in [Peri94]. The importance of using I/O characteristics of applications in processor scheduling is discussed in [Maju94].

2.6 Parallel I/O Management

In comparison to processor scheduling, much less work has been done in the area of management of parallel I/O on multiprogrammed systems. Characterization of file access patterns exhibited by parallel scientific workload is discussed in [Nieu95] [Kotz94]. These workloads characterize the usage of a multi-user file system consisting of many small requests. A comparison of different disk configurations that include disk striping, disk synchronization, for a transaction system workload as well as for a scientific application

workload, is presented in [Redd89]. The workload in [Redd89] represents single-user operations consisting of few requests for large amounts of data.

A comparison between two multiple disk management approaches, clustered and declustered, using three disk scheduling algorithms (FIFO, Shortest Seek First, Fastest Fitting Block First) is presented in [Livn87]. The declustered data in [Livn87] are distributed round robin style starting at a randomly selected disk and I/O request dispatching is handled centrally. The performance of three I/O request scheduling policies is considered in [Chen95] for RAID. One policy does not require information about current workloads on other devices and is thus suitable for distributed systems. The two other policies require knowledge about each of the devices in order to route requests to the device with the shortest queue or to minimize seek time. Scheduling I/O requests to reduce the total completion time of the I/O schedule given a limited set of resource pairs needed to service the requests is analyzed in [Jain93] as a minimization problem.

Optimal placement of data replicas in a distributed system to minimize communications costs has been studied by [Pres93]. Three schemes to replicate and distribute media objects of different media types are proposed in [Ghan93]. Each scheme results in a different number of replicas and different degree of mutual object interaction and I/O device contention.

2.7 Parallel I/O Management on Multiprogrammed Systems

The body of work on parallel I/O surveyed in the previous subsections has investigated

specific changes to hardware and the operating system to improve I/O performance, but has not explored what issues underlie the effective overall management of parallel I/O. The authors in [Jain93] have noted that an integrated approach at all system levels (application, system software, architecture) is needed to address the I/O bottleneck problem. In this spirit, the research for this thesis explores three issues important to parallel I/O management. These issues, and the motivation for selecting them, are listed below

1. **Application workload:** Because the purpose of a computing system is to run applications, understanding how I/O is performed by applications should facilitate understanding the system's behaviour and therefore the system's performance.
2. **Data distribution:** Research has shown that distributing data across many devices is beneficial because of the I/O parallelism and the distribution of I/O workload. Investigation of the different data distribution policies can lead to understanding their relationships with I/O parallelism and I/O workload distribution. This in turn can aid in selecting an appropriate distribution scheme to maximize system performance.
3. **Job based I/O scheduling:** Because an I/O device may receive requests from several concurrent parallel applications which do parallel I/O, the order with which the device services the requests can affect the performance of the I/O system. A study is necessary to understand the impact of various I/O scheduling policies on a system's performance. Where the size of data transfer of each request is large compared to the overhead of the access, alternative policies which do not schedule based on disk head movement may be appropriate. A number of such scheduling policies are investigated.

The I/O characteristics of applications and I/O management strategies such as data distribution has been reported by [Maju95] to have an impact on system performance. [Maju95] characterizes three models of applications which do I/O and computations (the NO, O1 and O2 models) and three data distribution strategies (Replicated-C, Replicated-D and Fixed) to show that certain types of applications and data distribution strategies can be significantly better than others.

Using [Maju95] as a starting point, the research presented in this thesis introduces a new I/O model (called NOP) and several new I/O management strategies such as device allocation policies (Random and Round Robin allocation strategies), data distribution policies (FixedR-C, FixedR-D, FixedRW(RR)) and job based I/O scheduling policies (JBNOT, JBOJF, JBRR, JBLOIO and JBSOIO) into the study of parallel I/O management.

Existing work on parallel I/O has focused primarily on environments that run a single application in isolation (see [Redd89] for a scientific application example). This thesis' research differs in that the focus is on running more than one application concurrently in the multiprocessor system in order to study the effects of inter-job resource contention caused by the multiprogrammed applications.

[Press93] considers optimizing the placement of replicas of a single object across a network with respect to minimizing communication costs of reading and writing data. When data replication is used in this thesis, the data is assumed to be replicated across all the devices and the communication costs are assumed to be negligibly small. In addition, data is replicated only for read operations and are not replicated for write operations. The repli-

cation and distribution of media objects in [Ghan93] is motivated by the desire to reduce device contention to maintain continuous retrieval of multimedia objects. The experiments in this thesis accept device contention caused by I/O requests from different jobs as an integral part of a multiprogrammed system and do not attempt to maintain continuous retrieval of information.

Studies which have used I/O scheduling algorithms to determine which request an idle device should service next have typically used algorithms such as Shortest Seek Time First (SSTF) which are formulated to reduce disk head movement (see [Livn87] for example). The I/O scheduling policies proposed in this thesis choose requests based on relative job priorities, with the priorities being determined using job related information rather than disk head location. This is motivated by high performance applications whose I/O requests have data transfer times much larger than the overhead of the data access such that scheduling algorithms no longer need to focus on minimizing the access overhead.

Knowledge gained from this research can be used in the design of operating systems engineered to support a general purpose multiprogrammed parallel environment. The design of such an operating system poses a number of high level design questions regarding the handling of different workload types by the operating system or the types of I/O management policies the operating system should provide. The questions include the following:

1. Applications perform I/O in various different ways. Do the I/O characteristics of applications have a strong relationship with I/O management and performance? Is there a clear winner among the different I/O models? Should an application be restructured to run differently to improve its performance? These questions are investigated for systems running a single job in isolation as well as running jobs concurrently.
2. Replicating input data for an application on a number of I/O devices may improve performance in a multiprogrammed system. The I/O requests for an application, for example, may be sent to any free I/O device on the system. What is the impact of the presence or absence of input data replication on performance?
3. Management of I/O may be performed in a number of different ways. A centralized management of I/O requests may be appropriate for small to medium scale parallel systems. On such a system, the status of each I/O device (idle/busy) may be stored in a centralized database. A decentralized approach is necessary for larger systems on which maintaining a centralized database may be too expensive. How do the distributed and centralized approaches to I/O management compare with respect to system performance?
4. Does the performance of a given I/O model depend on the way data is distributed on the different devices? How does the performance of data distribution strategies vary when the I/O management approach is changed from centralized to decentralized?

5. Given a certain type of application run on a system, what is an appropriate I/O scheduling policy to use? Does a centralized or distributed management policy affect this choice of scheduling policy?

A discrete event simulator is used to build an abstract model of a multiprogrammed parallel system, the I/O applications, the data distribution policies and the job based I/O scheduling policies (see Chapter 3 for details of the simulator). Such a model is appropriate for the high level questions addressed in this thesis. The simulation results are calculated to a 95% confidence level and values obtained generally are within 5% of the mean value. A factor-at-a-time approach is used to run the simulations. One parameter in the simulation model is varied while the others are held at fixed values. A large number of simulation experiments are run, but for the sake of conservation of space, only a subset of the experimental results is included in the text of this thesis.

Chapter 3 The System Model and System Model Simulator

This study uses an abstract model of a multiprogrammed system consisting of multiple processors and I/O devices. The system model consists of two components: a model of the processor & I/O subsystem (see Figure 1) and the workload model. The processor & I/O subsystem is described in this chapter and the workload model is described in Chapter 4.

FIGURE 1. High level diagram of the processor & I/O subsystem

3.1 The Processor & I/O Subsystem Model Description

The processor and I/O subsystems are characterized by the number of processors, P , and the number of I/O devices, I , respectively (see Table 1). Statistically identical (ie. of a single class or type) jobs enter and run in this open system. During its execution, a job performs computations in the processor subsystem and requests I/O from the I/O subsystem. When all the I/O and computations performed by a job are completed, the job exits the system.

TABLE 1. Parameters characterizing the system model

Parameter	Description
P	Number of CPUs in processor subsystem
N_p	Number of CPUs allocated to a job
$M = \text{ceiling}(P/N_p)$	Multiprogramming level (max number of concurrent jobs)
I	Number of I/O devices in I/O subsystem
d_s	Mean seek time of an I/O device
CV_s	Coefficient of variation of the device seek time
R	Mean job response time (queueing + execution times)

P identical CPUs in the compute subsystem are grouped into M processor sets (job nodes). Each set has N_p CPUs, but if P is not a multiple of N_p , then one of the sets will have only $P - N_p * (M - 1)$ CPUs. Each job that enters the system is sent to the first free job node. If no job node is idle, then the newly arrived job waits in a FIFO queue for the first idle job node. Static scheduling of processors, as in [Maju91], is used so once a job acquires a job node, all the processors in the node are held by the job until the job has terminated. Such a scheduling method is appropriate when a high overhead in switching processors between applications exist.

After a thread issues an I/O request, the thread is blocked waiting for the I/O request to finish. During this period, the CPU on which the thread was running becomes available for use by other threads in the same job. Processors may be shared by threads in an application in different ways; in this research, a ready thread is assumed to run on the first available processor.

An I/O request from a job is sent to one of the I/O devices in the I/O subsystem. Each request is characterized by a data transfer time (I/O demand) to represent the time needed to transfer a chunk of data. This I/O demand is a random variable characterized by a mean value and a coefficient of variation.

Every I/O device is identical and each device has seek times characterized by a mean seek time, d_s , and a coefficient of variation of seek time, CV_s . For a disk device, the value of d_s would include the mean rotational latency plus the mean track seek time. Each I/O device operates independently (as opposed to synchronously [Kim86]) and more than one

device may be concurrently active servicing multiple requests from a job. I/O devices in the I/O subsystem are shared by all jobs.

The performance measure of interest is the mean job response time, R . The response time of a job is defined as the difference between the time at which the last thread in the job completes and the arrival time of the job into the system.

Because the experiments conducted for this thesis focus on systems characterized by large computational workloads and high I/O transfer times, the mean and coefficient of variation of seek time are set to relatively small values. In the experiments, d_s is held at 1% of the mean CPU demand, d (which is discussed in the following chapter), and CV_s is set to 1.0. To maintain focus on the higher level issues of I/O management, overheads associated with scheduling and context switching are not considered. Although computation times and I/O transfer times for applications are modelled, messaging costs for maintaining a database of the status of I/O devices are not considered in the experiments. This is appropriate for small and medium scale shared memory or smaller distributed memory systems where the cost of communications are low so that maintaining a central database of device status is feasible. For larger distributed systems where communication costs are significant, maintaining such a centralized database may not be feasible and a decentralized strategy may be more appropriate.

The congestion on data channels between the I/O and compute subsystems are assumed to be negligibly small and no I/O buffering is employed. This is appropriate for the qualitative nature of the research that focuses on a number of basic issues that underlie

data distribution and scheduling.

Additional complexities such as media synchronization and real-time scheduling arising from scheduling I/O for multimedia applications [Gemm95][Redd94] are not addressed by this thesis. Additional constraints must be imposed on the scheduling algorithms in order to address these additional complexities. The scheduling policies presented in this thesis are targeted towards general purpose computing systems.

3.2 The Processor & I/O Subsystem Simulator Design Description

A computer simulation program models the abstract multiprogrammed system using the discrete event simulation software package called PARASOL [Niel95]. PARASOL provides the functionality (eg. random number streams, messaging etc) and components (eg. CPUs, links etc) needed to model and simulate a parallel, multiprocessor system. The simulation program makes calls to the PARASOL package routines to define and connect the components of the multiprogrammed system model. A description on how to run the simulation program is given in Appendix A. A high level description of the internals of how the multiprogrammed system model is constructed using PARASOL and how the simulation program is designed is presented in the remainder of this chapter. A diagram showing the PARASOL structure of the simulation program is given in Figure 2.

The Base Node automatically created by PARASOL with only one CPU is used to run the JobCreator(), the JobScheduler() and the IOScheduler() tasks. The JobCreator() task creates jobs at the specified job arrival rate and sends them to the JobScheduler() task. The

FIGURE 2. Internal structure of system simulator

JobScheduler() task accepts incoming jobs and assigns them to idle Job Nodes for execution. The IOScheduler() task accepts I/O requests from the Job Nodes and routes the requests to the I/O devices, plus it notifies the I/O requestor when its I/O request has completed. Because these three tasks run inside the Base Node which has only one CPU, when one task blocks, it must release the CPU in the Base Node so the CPU may be used by the other tasks in the Base Node. By following this rule, concurrent execution of the three tasks can be achieved with only one CPU in the Base Node.

Job Nodes are created by the simulation program. The number of Job Nodes corresponds to the multiprogramming level, M , of the system. Each Job Node is created with N_p CPUs, but if the total number of CPUs in the system is not a multiple of N_p , then one

Job Node will have fewer CPUs than the other Job Nodes. CPUs are the basic components in PARASOL for modelling entities that consume time, such as a computation thread.

When a job arrives in the system, the `JobScheduler()` task sends the job to the first idle Job Node. After a Job Node receives a job, the main control thread called `MainJobThread()` begins to run. If the job is limited to using a fixed set of I/O devices, then this thread assigns the devices to the job. `MainJobThread()` then performs its primary function, which is to mimic the I/O and computational behaviour of the job. In doing so, this thread may issue I/O requests and fork children threads. After forking the children, `MainJobThread()` blocks until all the children threads have completed, at which time it resumes running. When `MainJobThread()` finishes mimicking the job, it notifies the `JobScheduler()` task that the Job Node is again idle.

The children forked by `MainJobThread()` all run instances of the `ChildThread()` thread on the Job Node's CPUs. `ChildThread()` can issue I/O requests and perform computations. During an I/O operation, `ChildThread()` blocks and releases the CPU on which it was running. As each child terminates, it notifies `MainJobThread()`.

The threads of a job within the Job Node can send I/O requests to the I/O subsystem. These requests are received in the I/O subsystem by the `IOScheduler()` task which routes the I/O requests to I/O devices according to the routing scheme for the data distribution policy that is in effect. For fixed device I/O, the I/O request is immediately sent to the specified I/O device. For distributed routing, `IOScheduler()` task randomly chooses a device and sends the I/O request to that device. For centralized routing, `IOScheduler()`

task queues all I/O requests until an idle device is found, at which time a request is taken off the queue and sent to that idle device. Queueing structures used for implementing the job based I/O scheduling policies are maintained by IOScheduler() task which also implements the algorithms for each of the I/O scheduling policies.

One I/O Device Node is created by the simulation program within which is one CPU for each simulated I/O device in the system. An instance of the IODevThread() thread runs on each CPU to simulate the seek and data transfer times for an I/O device. This thread is usually blocked waiting for I/O requests from the IOScheduler() task. After servicing an I/O request, this task notifies the IOScheduler() task of the completion by sending a message. If more I/O requests are waiting for servicing by that I/O device, then IOScheduler() sends the next request to that device's IODevThread().

Several major data structures are globally accessible by the simulation code to store the important data used during a simulation run (see Figure 3). The input parameters for a simulation run are stored in three data structures called 'phys_system[]', 'sim_parms[]' and 'iom_parms_table[]', corresponding to the physical system, the simulation and the job model parameters respectively. Information about each Job Node such as what it is running is stored in the data structure called 'jnode_table[]'. Information about each I/O device is stored in the data structure 'iodev_table[]'.

When a centralized data distribution policy such as Replicated-C is in effect, the linked list of I/O requests queued waiting for an idle I/O device are pointed to by the head pointer 'waitingreqs'. When a distributed or fixed policy is in effect, the linked list of queued I/O

FIGURE 3. Data flow diagram of the simulator

requests waiting for an idle I/O device are pointed to by the head pointer in the 'reqlist' field within the 'iodev_table[]' structure. I/O requests waiting for an idle I/O device are taken either from 'waitingreqs' or from 'iodev_table[]' according to the type of routing (Centralized or Distributed/Fixed). For data distribution policies which use both centralized and distributed (eg. FixedR-C) routing, the IOScheduler() task first checks for requests in 'iodev_table[]'. If the table has waiting requests, they are sent to the idle device. If 'iodev_table[]' does not have requests waiting for the device, then IOScheduler() task checks for requests in 'waitingreqs' and sends those requests, if any, to the idle device.

Procedure variables are variables which hold a reference (ie. an address) to a function. Such variables are used extensively in the simulation program. Based on the input parameters to the simulation program, the desired functions are bound into procedure variables which are later called during the course of the simulation run. The use of procedure vari-

ables provides flexibility to the simulation program to support different job models or policy options at run time. Procedure variables are used to implement the various I/O job models, the device allocation policies and the queueing routines for the I/O scheduling policies.

The two-dimensional table 'run_job[]' implements a finite state machine for each job model type. Each row of the table corresponds to a specific job model while each column corresponds to the job model's current execution state. Although a one-dimensional array would have been adequate, 'run_job[]' is a two-dimensional array to allow more than one job model type to run in the system in the future. The state corresponds to the particular stage at which the job model is currently executing. At each stage of execution, a procedure for the corresponding job model is invoked to carry out the actions appropriate for that job model. As an example, if the I/O workload were NOP jobs (see Chapter 4 for description), then the set of procedures called to simulate the NOP would be PreForkNOP(), ForkAndJoinDef(), RunChildDef() and PostJoinNOP() for the pre-children-thread-fork, the forking-of-children-thread, running-of-children-thread-code and post-children-thread-join stages respectively.

Device allocation strategies are implemented using the four procedure variables 'InitDevAllocPV', 'PreDevAllocPV', 'DoDevAllocPV' and 'PostDevAllocPV' that correspond to one-time initialization of any data structures done at system initialization time (if required), processing prior to job start, the actual determination of what device to allocate to a thread or operation and lastly, the cleanup work prior to job termination, respectively.

As an example, if Round Robin device allocation were used, then the procedures bound into these variables would be NULL, PreDAProcRR(), DoDAProcRR() and NULL respectively. The NULL is not a procedure address, but is a value meaning nothing needs to be done at that stage.

The table 'SchedGetDev[]' is a two dimensional table of procedure variables indexed by the I/O operation (read or write) and the data distribution policy. The table entries point to the I/O device assignment functions corresponding to the desired data distribution policy for the desired I/O operations. For the Replicated-C policy, for example, both the read and write operations result in calls to the function GetDevAnyC(), while for the Replicated-D policy, the read and write operations result in calls to the function GetDevAnyD().

The table 'EnqElemProc[]' contains pointers to procedures to enqueue an element onto one queue. This permits implementation of enqueueing policies, such as FIFO, or smallest request first or largest request first. If all elements were to be enqueued on a FIFO basis, for example, then the procedure EnqElemFIFO() would be invoked whenever an I/O request is to be put onto a waiting queue.

The table 'DeqReqProc[]' is indexed by the specific job based I/O scheduling policy to dequeue an I/O request according to the I/O scheduling policy. The procedure that is bound for each scheduling policy determines which job with waiting I/O requests has the highest priority. Once that is determined, then an I/O request is removed from the head of queue corresponding to that job. Note that the enqueueing operation must order all elements in a queue such that the element at the head of the queue is the next element to take

during the dequeue operation. If, for example, the policy JBOJF is in effect, then the procedure `DeqReqFIFO()` would be invoked to look for an I/O request from the oldest job.

Statistic variables of interest, such as the job response time, are defined by the simulation program. Sample values of the variables are passed to PARASOL which calculates the mean and the confidence intervals for the variables. The lengths of simulation runs are set to large enough values for PARASOL to produce a 95% confidence level and values that are generally within 5% of the mean value.

Chapter 4 Parallel I/O Job Models

Four simple I/O job models are used to represent parallel applications applied as the workload into the multiprogrammed system. These models all use a fork and join architecture modified to include I/O operations. Job models with fork and join architectures have been used in various studies on resource management on parallel systems (see [Leut90] for example). The I/O job models used in this thesis are motivated by several I/O characteristics observed by previous studies which characterized I/O patterns of uniprocess and multiprocess scientific applications running on a multiprocessor, parallel environment. The applications monitored by [Pasq94a] often alternate computation with I/O, and that the duration of each I/O burst is often much shorter than the duration of computes. The studies by [Kotz94] [Nieu95] [Pura95] showed that processes from a multiprocess application do I/O independently of the other processes in that application; that individual I/O request sizes tended to be relatively small; that little data sharing occurs between processes within a job and no data sharing happens between jobs.

The four simple I/O job models, called the NO (No Overlap), O1 (Overlap #1), O2 (Overlap #2), and NOP (No Overlap-Parallel), all have a fork and join architecture (see Figure 4) but they differ primarily in the amount of I/O parallelism and when and where that I/O is done. The NO model does not do parallel I/O while NOP, O1 and O2 do I/O in parallel. A NO or NOP job cannot be simultaneously doing I/O and computation (no CPU-I/O overlap) while an O1 or O2 job can.

The NO, O1 and O2 models have been used in a previous study of I/O management

[Maju94] but the NOP model is new and is introduced in this thesis. These four simple models are appropriate for the high level questions investigated in this thesis plus they also reflect the behaviour of certain classes of programs executing on real systems. Jobs characterized by the presence and absence of CPU-I/O overlap have been observed in real parallel computing that include scientific code and graphics software [Cyph93]. Similar observations for parallel scientific workload are reported in [Kotz94]. Each of these models is described in more detail below.

The *NO Model* (see Figure 4a): A NO job begins by first reading in some data using a single read operation from a single I/O device. When the read is done, the job forks into n_d 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 can correspond to the writing of the results of the computation. The job terminates when its I/O completes. This job model does not have CPU-I/O overlap because the job can only be either doing I/O or computation, but not both simultaneously.

The *NOP Model* (see Figure 4b): A NOP job begins like a NO job by first reading in

some data. The read, however, is done using n_i parallel I/O operations. When all these read operations have completed, the job forks n_d children threads, each of which only performs computations. When all these threads complete their computations, the job performs n_i parallel write operations. After all n_i write operations are completed, the job terminates. Like the NO model, the NOP also has no CPU-I/O overlap.

The O1 Model (see Figure 4c): This represents jobs which overlap I/O with computations. A job starts by forking n_d children threads, each of which reads its input data from an I/O device and then performs some computation. After completing its computation, each thread writes the results onto an I/O device. When all n_d children threads have completed their writes, the job terminates. This job model has CPU-I/O overlap because I/O being performed by a thread can overlap with the computation being performed at the same time by another thread of the same job.

The O2 Model (see Figure 4d): This is similar to the O1 model but has more I/O-computation cycles per thread and has more overlap between I/O and computations. Each of the n_d children threads begin by reading input data from an I/O device. Each thread then alternates between doing computations and doing I/O (either reads or writes according to a read-write ratio). The last operation a child thread does is to write some data onto a device. When all n_d threads have completed their last write, the job terminates. Like the O1 job model, this job model has CPU-I/O overlap.

The NOP is similar to the O1 job model in that both can do parallel I/O. A NOP job, however, must complete all its read I/O operations before it can start its computations, and

all its computations must complete before the job can start its write operations. An O1 job does not have this restriction because a child thread can do its I/O or computation independently of the job's other child threads so an O1 job can be simultaneously doing I/O and computation. Additionally, a NOP job can have more parallel I/O operations, n_i , than child threads, n_d , whereas an O1 (and O2) job is constrained to have $n_i = n_d$.

Each job model has a pattern to the I/O requests it sends to the I/O subsystem. A single instance of a job model running alone on a system with an unbounded number of processors, P , and an unbounded number of I/O devices, I , has the corresponding I/O pattern illustrated in Figure 5. The figure shows the amount of data that is transferred from the job as seen by the I/O subsystem over the time the job runs. NO and NOP jobs transfer data in two distinct bursts. NO jobs send only one request but NOP jobs send n_i requests for each burst. O1 and O2 jobs both initially send n_d read requests. O1 jobs subsequently send n_d while O2 jobs send $(k-1) \cdot n_d$ read or write requests spread out over time.

FIGURE 5. I/O patterns of different job models

The storage of data retrieved from the I/O subsystem by a job model may endear certain job models to certain types of computing systems. Because the data retrieved by an O1 or O2 child thread are used by the child thread itself, these two job models may be suit-

able for both shared and distributed memory systems. Because the number of I/O operations in NOP and NO job models can be different from the number of children threads in the model, the retrieved data might best be stored in a shared memory environment to simplify data access by the children threads. An additional complexity of distribution of the retrieved data may arise when NOP or NO job models are used in a distributed memory system.

If the I/O subsystem consists of RAIDs, then certain job models may be more suitable to a particular RAID configuration in the subsystem. If the disk array is a level 1, 2 or 3 RAID, then NOP jobs running in isolation is a good match because all the disks in the array group are involved in servicing all the parallel I/O operations in a NOP I/O burst. For a multiprogrammed system running NOP jobs, however, the involvement of all the disks in a group to service one I/O burst can cause undue delays to other NOP jobs wanting to do I/O. The ability for levels 4 and 5 RAID configurations to service multiple requests simultaneously make them more suitable for O1 and O2 jobs' I/O patterns. These two configurations are also better for handling NOP jobs running either in isolation or concurrently with other jobs in the system.

The workload the system is subjected to consists of applications characterized by any one of the job models. The four job models can be characterized using a common set of parameters (Table 2) plus the mean CPU execution time (CPU demand) per CPU request and the mean I/O execution time (I/O demand) per I/O request. Jobs arrive into the processor subsystem as a Poisson process characterized by an arrival rate λ . The number of chil-

TABLE 2. Common set of workload model parameters

Parameter	Description
λ	Mean arrival rate of jobs.
n_d	Number of children threads in a job.
n_i	Number of parallel read or write I/O operations of a NOP job.
d	Mean CPU demand of a child thread.
CV_d	Coefficient of variation of CPU demand for a CPU request.
k	Number of I/O operations per O2 child thread ($k \geq 2$).
p	Probability that an O2 job's I/O request is a read.
F_i	I/O factor.
CV_i	Coefficient of variation of I/O demand of an I/O request.

children threads and the number of parallel I/O operations are represented by n_d and n_i , respectively. The mean CPU demand of a child thread is d and the coefficient of variation of a CPU request is CV_d . The probability that an I/O operation done by an O2 job's child thread is a read is given by p , while the number of I/O operations per O2 child thread is k . The coefficient of variation of an I/O request is CV_i . The mean total I/O demand of a job (IOD) represents the mean total I/O workload performed by a job. The total I/O workload is the sum of the I/O workload from each I/O operation in a job. An important characteristic of parallel applications introduced in [Maju94] is the I/O factor which expresses the IOD as a fraction of the mean total CPU demand of the job ($n_d * d$):

$$IOD = n_d * d * F_i$$

The CPU execution time of each CPU request is an independent, identically distributed random variable with mean value given by the column "Mean CPU Demand per CPU Request" in Table 3 and a coefficient of variation given by CV_d . The I/O execution time of each I/O request is also an independent, identically distributed random variable with a mean value given by the column "Mean I/O Demand per I/O Request" and a coefficient of

variation given by CV_i .

TABLE 3. Mean I/O and CPU demands per request for different job models

Job Model	Mean CPU Demand per CPU Request	Mean I/O Demand per I/O Request
NO	d	$IOD/2$
NOP	d	$IOD/(2 \cdot n_i)$
O1	d	$IOD/(2 \cdot n_d)$
O2	$d/(k-1)$	$IOD/(k \cdot n_d)$

All the parameters such as response times and CPU or I/O demands are normalized with respect to the mean CPU demand of a child thread, d . If ' d ' is expressed in units of seconds, then quantities such as R would also be in units of seconds. Since the coefficient of variation of random variables used in the simulation experiments are greater than or equal to one, random value generation is based on a bi-phase hyperexponential distribution [Koba81] (see Appendix B). Similar techniques for random number generation have been used by other researchers such as [Leut90]. The performances of the job models are investigated both in the context of a single application running in isolation as well as in a multiprogrammed environment. The results are presented in the following subsections.

4.1 Performance of a Single Job Running In Isolation

Each of the four I/O job models have different parallel I/O characteristics. How this difference affects a particular job model's response time in the presence of I/O resource contention is investigated by varying the number of I/O devices available to a job as a means to introduce intra-job I/O resource contention. All the available processors and I/O devices are used by that job. To eliminate inter-job resource contention effects, the job models are run in isolation ($M=1$). A read-anywhere-write-anywhere I/O management strategy where

all I/O requests are routed to idle devices using a central router is used in this experiment (a more detailed description for this policy, the Replicated-C policy, is provided in the next section). The results for $F_i=2.0$ are shown in Figure 6 for the four job models.

The results of the experiment shown in Figure 6 demonstrate a large performance improvement is possible by parallelizing the I/O performed by an application when using multiple I/O devices. Because a NO job only has one I/O operation active at any one time, the mean job response time is invariant with respect to the number of I/O devices in a uni-programmed environment. For the NOP, O1 and O2 jobs, the mean job response time drops dramatically when the first few devices are added to the system, but the incremental performance gain soon decreases until R becomes insensitive to the addition of any more

devices. The values of I at which R becomes somewhat insensitive to the addition of more I/O devices are less than either n_i or n_d . Because each I/O request's demand is variable (some big, some small), devices servicing the small requests becomes idle sooner than the devices servicing big requests. A device that becomes idle is reused by the central router to service any waiting requests so fewer I/O devices than n_i or n_d may be adequate to service a job's parallel I/O operations.

Decreasing F_i reduces the value of I at which R becomes insensitive. At the same time, the difference between $O1$ and $O2$ at high I decreases but both $O1$ and $O2$ are still better than NOP (see Figure C-1 in Appendix C for example).

Figure 6 also shows that for a given amount of I/O parallelism, better job performance may be obtained by jobs which have CPU-I/O overlap. The experiment uses NOP and $O1$ jobs having the same number of parallel I/O operations ($n_i = n_d$) but the mean job response time for $O1$ is less than for NOP at low and high values of I . The mean job response times for NOP jobs can be decreased by increasing n_i , the I/O parallelism (see Figure 7). At large I and n_i ($I=20$, $n_i=40$, $n_d=10$ in Figure 7), the mean job response time of NOP is seen to be slightly less than the $O1$ time ($I=20$, $n_d=10$ in Figure 6). This improvement in NOP performance can be attributed to the increase in the NOP I/O parallelism when n_i is increased whereas the I/O parallelism for $O1$ is restricted by the number of children threads, n_d . Decreasing F_i reduces the gains NOP makes on $O1$ at high I and n_i . When $F_i=0.5$ (and $I=20$, $n_i=40$, $n_d=10$ as above) the NOP mean job response time just matches the corresponding $O1$ time (see Figure C-2 in Appendix C).

The case of $I=1$ is a degenerate case where increasing the number of parallel I/O operations actually degrades performance. This degradation is likely caused by the increase in the number of seeks done by the I/O devices to service the larger number of I/O requests, n_i , from the job. As n_i increases, the mean I/O demand of each request becomes smaller and the proportion of seek overhead per request as well as the total seek overhead for the job's I/O increases. If the system has insufficient number of I/O devices (eg. $I=1$) across which to spread the n_i requests, these n_i requests and their associated seek overhead causes the job response time to increase. This effect is observed also for $I>1$ (eg. $I=2$, and $n_i>35$) although the effect is more noticeable when the I/O demand per request is smaller such as when F_i is small (see Figure C-2 for $I=2$ and $n_i>20$ in Appendix C). Interestingly, the graph in Figure 7 for $I=20, n_i>20$ suggests that limited improvement in mean job response

time still can be gained by increasing n_i beyond 1. This appears to indicate the system has sufficient I/O devices to offset the increase in seek overhead caused by increasing n_i to 40.

4.2 Performance in a Multiprogrammed Environment

Multiprogramming applications together introduces inter-job resource contention effects not present in a uniprogrammed environment. How the mean job response times of the different job models with their different I/O characteristics are affected by the presence of this contention is studied in this section. The degree of contention for the I/O devices can be varied by varying the job arrival rate which affects the loading of the system. Figure 8 plots the mean job response times obtained with the NOP, O1 and O2 job models as a function of λ . NO model was found to perform poorly in [Maju95] and is omitted as a function of λ .

from this experiment. The same write-anywhere-read-anywhere centralized routing policy used in the unprogrammed experiments is also used for this experiment.

For a given number of parallel I/O operations in a job, the NOP model, which does not have CPU-I/O overlap, performs poorly compared to O1 and O2, both of which have CPU-I/O overlap. Without CPU-I/O overlap, the CPUs for a NOP job are idle when I/O operations are in progress and vice versa. This unnecessary idle time leads to longer mean job response times for NOP jobs. The mean job response time of O2 jobs increase sharply above O1 at medium to high λ . To fully understand why O2 performs worse than O1, further research is needed, although the cause might be related to the larger number of small I/O operations done by O2 to move the same average amount of data as O1. This O2 behaviour is in contrast to the unprogrammed results which showed O2 to be comparable to O1 when the system has little resource contention.

At higher F_i , the O2 mean job response times exceed those of O1 at a low value of λ , and the times even exceed those of NOP. Figure C-3 in Appendix C shows the results for $F_i=2.0$. The results suggest that for moderate to high values of λ , too many small I/O requests and therefore too many seek operations by the device can be detrimental to the performance of O2 type jobs especially when F_i is high. These results are again different from the unprogrammed results where at high F_i , O2 mean job response times are slightly lower than O1 when the system has little or no contention.

In the unprogrammed experiments, increasing n_i and I can substantially reduce the mean job response times of NOP jobs to the point that NOP is comparable (at low $F_i=0.5$)

or even slightly better (at higher $F_i=2.0$) than OI. Increasing n_i and I to 40 and 20 respectively while keeping $n_d=10$ and $F_i=0.5$ in the multiprogrammed case shows the job response times of NOP jobs remain slightly greater than OI as shown in Figure 9.

Increasing I brings about the expected decrease in R for both OI and NOP (compare Figure 8 and Figure 9). Increasing n_i also brings about the expected decrease in R for NOP, but even at $n_i=40$, the mean job response times for NOP remain slightly larger than for OI with $n_d=10$. At higher $F_i=2.0$, the mean job response times for NOP with $n_i=40$ are seen to be less than for OI as shown in Figure 10. These results show that a larger number of I/O operations and I/O devices can help NOP performance approach and even exceed that of OI in a multiprogrammed environment when the job is I/O intensive.

that of O1 in a multiprogrammed environment when the job is I/O intensive.

4.3 Discussion

Experiments with different I/O models have led to some general observations which appear to be applicable in both uni- and multiprogrammed environments regarding the behaviour of jobs with certain I/O characteristics. Jobs which perform parallel I/O have significantly lower job response times than jobs like the NO model which do not perform any parallel I/O. Increasing the number of concurrent I/O operations such as for the NOP model can lower job response times but having too many NOP parallel I/O operations to move a given amount of data might actually increase the response time if the access overhead becomes significant. Job models which have CPU-I/O overlap such as the O1 can

such as NOP coupled with a higher I/O factor can lower NOP mean job response times below those of an O1 model.

A job model which has a large number of I/O operations interleaved with computations, such as the O2, can given slightly better response times than an equivalent O1 job when single jobs are run in isolation. When jobs are multiprogrammed, the effects of many smaller interleaved I/O requests become more complex and may even deteriorate O2 performance compared to O1 or NOP.

Chapter 5 Data Distribution Policies

Three classes of data distribution strategies are investigated: Write Anywhere Read Anywhere ("Replicated"), Fixed Read and Write ("FixedRW") and Fixed Read ("FixedR"). The Replicated and FixedR classes are subdivided into Centralized (Replicated-C and FixedR-C) and Distributed (Replicated-D and FixedR-D) approaches. In the Centralized approach, a centralized dispatcher monitors the status of all devices (busy or idle) and a waiting I/O request is sent only to an idle device. In the Distributed approach, an I/O request is automatically sent to a device without regard to or knowledge of the current device status (busy or idle). The Centralized approach 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 system, a high cost is associated with maintaining such a centralized database so a decentralized approach for I/O management such as the Distributed approach may be more appropriate.

The Replicated class of data distribution policies allows any device to service a read or write I/O request. To allow a job to read from anywhere, the read data is replicated across all the I/O devices. Write I/O requests can be serviced by any I/O device and only a single copy of data is written by an application to the I/O devices. Not replicating write data would be suitable for scientific applications where the output results are written to I/O devices and this data written by one thread are not needed by the other threads in the application.

The FixedRW class of policies uses specific I/O devices to service a read or write I/O request. This means no replication of data is possible. The FixedR class of policies uses a specific I/O device to service a read I/O request, but write requests can be serviced by any I/O device. Because read requests are sent to specific devices, no read data replication is needed. Write data also is not replicated for this class of policies.

The FixedRW and FixedR classes of policies are studied to see if motivation exists to replicate data like that done by the Replicated class. Because write data is not replicated for FixedR class, this class is studied to see if relaxing the restriction that a specific device can service a particular write operation will give better performance than FixedRW.

To illustrate how these three classes of data distribution policies route I/O requests, a conceptual diagram of the Replicated and FixedR policies is shown in Figure 11. The FixedRW policy diagrams are similar to FixedR-D except that the write requests are handled by a fixed router rather than a distributed router. Detailed descriptions of how each of the data distribution policies handle read and write I/O requests received by the I/O subsystem follows.

In *Replicated-C* (Figure 11a), all I/O requests received by the I/O subsystem are inserted into a central queue. A central router which maintains the busy/idle status of every I/O device sends an I/O request to the first idle I/O device the router finds. If no device is idle, then all requests remain in the queue until a device becomes free. In the *Replicated-D* policy (Figure 11b), no central database of device status is maintained so each request is sent immediately to one of the I/O devices chosen at random with proba-

bility $1/I$. At the device, the request is put into a queue maintained and accessed only by the local I/O device. If the device is idle, then the request is processed immediately.

The three I/O routing policies proposed by [Chen95] (which are called I/O scheduling policies) are strictly for routing read requests while write requests are routed to all the devices which have copies of the data. The Random Join policy routes read requests to a randomly selected device. The other two policies route read requests to device with the shortest request queue (Shortest Queue policy) or to the device whose last queued request's cylinder address is closest to the new request's address (Minimum Seek policy). The Random Join is similar to the routing done by the Replicated-D policy but the random routing in Replicated-D is applied to both read and write requests. The other two policies use a centralized dispatcher to route read requests to satisfy the desired criteria. The central dispatcher used by the centralized policies in this thesis differs in that it routes both read and write requests and only to idle devices.

For the *FixedR* class of policies, all the read requests are sent immediately to the specified device which has the desired data, regardless of the current device state. The read requests are put into a queue local to and maintained by the device. With the *FixedR-C* policy (Figure 11c), write requests received by the I/O subsystem are put into a central queue from which a central router which has access to the device status database routes a write request to an idle device. A read request on the other hand is routed to the target device as soon as it is generated. When a device becomes idle, it first checks its local queue for more read requests to process. In this way, the read requests are served by the device immediately after the completion of the current request, and the handshaking between an idle device and the central router is avoided for read requests. If the local queue is empty, the idle device notifies the central router and accepts a write request sent from the central router.

With the *FixedR-D* policy (Figure 11d), write requests are sent immediately to a device chosen at random with probability $1/I$, regardless of the current device state, because any device can service the write requests. This is not the case for read requests because they can be serviced only by specific devices. Under *FixedR-D* policy, both the read and write requests wait on a queue local to the device if the device is busy. When the device becomes idle, the request at the head of the queue is removed from the local queue for servicing.

The *FixedRW* class of policies sends all read and write I/O requests immediately to the specified I/O devices regardless of the current busy/idle state of the device. At the device,

requests are put into a queue maintained and accessed by the local I/O device. If the device is idle, it takes and services requests from this local queue.

The specific set of devices used by a job to service its I/O requests when the FixedR and FixedRW policies are in effect are assigned (allocated) to that job when it begins to run and is allocated from the set of all available devices in the system. The set of allocated devices correspond to the devices onto which read data has been distributed apriori, and onto which write data will be written by the job. For FixedR and FixedRW, the devices are allocated in one of the two following ways: Round Robin and Random. Note that the policy called Fixed in [Maju95] is called FixedRW(Rand) in this thesis.

With Round Robin device allocation (or equivalently, Round Robin data distribution), the maximum number of different I/O devices is allocated to a job to enhance its I/O parallelism. This can be expressed more concisely by stating that $\min(I, n_i)$ different devices are allocated to a NOP application and $\min(I, n_d)$ different devices are allocated to an O1 or O2 application. Data for the first I/O operation is assumed to be placed on device number 'k' chosen at random. The data for the remaining I/O operations are assumed to be placed on device numbers 'k+1', 'k+2' and so on in a round robin fashion. When all the devices have been chosen once, the devices may be reselected again if the job requires more devices than are available in the system. See [Jada95] for an application of this method of data distribution.

Such an organized allocation of input data to maximize the parallelism in I/O for an application may not always be possible. On a general purpose multiprogrammed system in

which the amount of data read from or written onto different I/O devices by different applications can vary substantially, imbalance in data placement may result. Certain devices may contain more data needed by applications than other devices which may lead to bottlenecks in I/O because an application may end up being restricted to using a smaller number of I/O devices than ideally possible. This scenario is investigated with the help of Random device allocation (data distribution) which allocates the set of devices to a job by selecting devices at random. Some devices can be chosen multiple times while some devices are not chosen at all because of this randomness. The reason for studying these two device allocation policies is to find out whether or not motivation exists to reorganize the stored data from time to time so that the Round Robin allocation can be used.

If the I/O subsystem consists of RAIDs, then certain data distribution policies relate well to certain types of RAID configurations. Policies which access specific devices only (eg. FixedRW) can be used with levels 1, 2 and 3 configurations because the devices used by a job are fixed and these devices can be organized into one array group. Concurrently executing jobs may encounter delays, however, if one array group is needed at the same time by more than one job. A level 4 or 5 configuration can exploit the flexibility of allowing any device to service a request under a Replicated policy. A level 1, 2 or 3 configuration coupled with the Replicated policies, however, puts all available devices into a single array group which may result in the I/O subsystem servicing one job at a time.

Two job models are chosen as the workload to apply to the system in order to study the relative performances of the data distribution policies. Only two are chosen to reduce the

experimental parameter space, and therefore, the number of experiments. The two models chosen are the NOP and OI models. They are chosen to represent two classes of jobs which do I/O in parallel: one which has no CPU-I/O overlap, and one which does have CPU-I/O overlap. For a given number of parallel I/O operations, jobs having CPU-I/O overlap were shown to perform better than jobs which do not have the overlap.

In the simulation experiments, a number of parameters are kept fixed. To maintain the focus of this study on I/O, CV_d is fixed at 1.0 and n_d is fixed at 10 for the job models. The multiprogramming level is chosen such that more than one job can run concurrently in the system (ie. $M > 1$), but each job should have multiple CPUs (ie. $M < P$) to permit a reasonable degree of computational parallelism. A value of $M=4$ seems to be a reasonable choice. From Figure C-1 in Appendix C, more than four devices do not appear to reduce the mean job response times significantly for the given set of experimental parameters so I is fixed at 4 for the experiments.

5.1 Relative Performances of Device Allocation Policies

Because the set of devices allocated to a job using the Random device allocation policy may include the same device multiple times, this allocation scheme can potentially induce extra intra-job contention for I/O devices on top of contention caused by insufficient number of devices in the system and external inter-job contention. To illustrate the effects of this extra intra-job contention caused by Random allocation, the mean job response time is plotted as a function of the number of I/O devices for Random and Round Robin allocation (Figure 12) using NOP workloads. Similar results are obtained for OI workloads (see

Figure C-4 in Appendix C). With a single device, the choice of the device allocation policy should not matter and the mean job response time for both policies should be identical, provided the system does not saturate first. When the number of I/O devices is unbounded, the list of devices randomly allocated to a job is unlikely to have the same device more than once so no intra-job contention should occur. This would be reflected by the two device allocation policies having the same asymptotic R for sufficiently large I . Performance differences between these two policies is seen to be greatest when the system has small number of I/O devices.

Because for a given number of I/O devices the performances of Random versus Round Robin allocation can differ, the extent of this difference is next investigated as a function

of the job arrival rate λ to characterize the behaviours of these two strategies when system loading increases. The results in the context of the FixedRW policies are shown in Figure 13 for OI workloads and Figure 14 for NOP workloads.

Two interesting observations can be made from the results for NOP and OI. First, at low arrival rates (ie. the job essentially is running in isolation), the performance difference between these two allocation schemes is non-zero. Second, this finite difference increases as λ increases. The Random allocation policy is seen to increase the mean job response times at high λ faster than Round Robin, particularly for NOP workloads.

The experimental results highlight the benefits that can be gained by letting a job use all the devices available in the system rather than limiting the job to using a subset of the

available devices. Random allocation can potentially reduce the parallelism of I/O for a job because even though the system may have enough devices, this allocation scheme can select some devices multiple times while not selecting devices at all for use by a job. This causes the job's I/O workload to be concentrated on only a few devices instead of spreading the workload across all the devices.

In contrast, Round Robin distributes data across as many devices as possible so that a given job is able to make use of as many devices as possible in parallel for its I/O. With this scheme, a device is selected more than once only if the number of devices required by a job exceeds the number of devices in the system. By spreading the I/O workload across many devices, Round Robin device allocation can reduce the time an I/O request must wait before being serviced by an idle device and hence reduce the job's response time.

Note that for NO jobs, the choice of Random or Round Robin device allocation policy (FixedRW(Rand) or FixedRW(RR)) is irrelevant because this model only has one read operation and one write operation so that only one device is needed at any time to do either the read or write.

5.2 Relative Performances of Data Distribution Policies

The data distribution policies are investigated by studying the increases in mean job response times caused by the policies as λ , CV_i and F_i are increased while holding other variables constant. Increasing λ introduces inter-job contention for the shared I/O devices by increasing the probability of multiple jobs running concurrently in the system. Increasing CV_i adds variation to the I/O demand of each request. Increasing F_i adds more workload to the I/O subsystem by increasing the mean amount of total I/O done by a job. Both Random and Round Robin device allocation strategies are employed in the experiments.

The distribution policies are studied using only the O1 and NOP job models as the workloads to reduce the parameter space of the experiments. These two models are used to represent jobs which have and do not have CPU-I/O overlap, respectively. The presence of CPU-I/O overlap was seen in the previous chapter to improve mean job response times.

5.3 Results for O1 workloads

Effects of Job Arrival Rate (λ):

At very low λ , the system has very little contention between jobs so the mean job response times of the different data distribution policies show little difference. As λ

increases, differences between data distribution policy performances begin to appear. With Random device allocation (Figure 15), the FixedRW(Rand) policy performs poorly at

moderate to high λ , with mean job response times surpassing the Replicated-D and FixedR-D(Rand) times. This may be because the Replicated-D policy chooses a set of devices twice (once for the reads and once for the writes), which increases the likelihood that the I/O load is spread across more different devices than FixedRW(Rand), which uses the same set of devices for both reads and writes. The FixedR-D(Rand) policy gives mean job response times very close to that for Replicated-D. This is expected because both the read and write devices are assigned randomly in the simulation. Even with Random device allocation for its read operations, the FixedR-C(Rand) policy shows that it can perform as

well as the Replicated-C policy.

Changing the device allocation policy to Round Robin (Figure 16), FixedR-C(RR) pol-

icy again gives comparable job response times to Replicated-C. The policy affected most by this change in allocation policy is FixedRW(RR) which shows a dramatic improvement over the Random allocation results as it now performs better than both FixedR-D(RR) and Replicated-D. FixedR-D(RR) policy performs better than Replicated-D at low and high λ largely because it uses Round Robin allocation to spread the read I/O workload across the devices. This compensates for the poor write I/O workload distribution caused by random device selection.

Effects of Coefficient of Variation of I/O Demand Per Request (CV_i):

When Random device allocation is used, increasing CV_i causes the FixedR-C(Rand) policy to diverge and become inferior to Replicated-C (Figure 17). Interestingly, the mean job response times using FixedRW(Rand) gradually improve to become better than both the Replicated-D and FixedR-D(Rand) policies.

When Round Robin device allocation is used, FixedR-C(RR) policy is observed to also diverge from and become inferior to Replicated-C as CV_i increases (Figure 18). FixedR-C(RR) continues to perform better than the FixedRW or Replicated-D policies. FixedRW(RR) policy also continues to perform better than the FixedR-D(RR) or Replicated-D policies. Because of the Round Robin device allocation of its read I/O requests,

FixedR-D(RR) performs better than Replicated-D.

The behaviour of FixedR-C(Rand) and FixedR-C(RR) policies as CV_i is increased can be attributed to the poor read I/O workload distribution caused by the fixed allocation of devices to each job for which even the centralized management of the write requests cannot compensate. Even with Round Robin allocation, requests still may be sent to busy devices, resulting in I/O service delays for the request. These delays are magnified when requests with large I/O demands enter the I/O subsystem (such as when CV_i is high) and force smaller requests to wait behind them. In contrast, Replicated-C is able to maintain good I/O workload balance across the devices even as CV_i increases.

Effects of Job I/O Factor (F_i):

At unity CV_i , for Random device allocation and increasing F_i , the behaviours of the data distribution policies is shown in Figure 19. The FixedRW(Rand) policy has slightly

larger mean job response times than FixedR-D(Rand) and Replicated-D. FixedR-C(Rand) performs comparably with Replicated-C over low and high F_i values. Both FixedR-C(Rand) and Replicated-C perform better than Replicated-D or FixedR-D(Rand).

At higher CV_i (Figure 20), FixedR-C(Rand) degrades in performance relative to Replicated-C with increasing F_i . FixedRW(Rand) performance improves to become better than Replicated-D and FixedR-D(Rand). This is to be expected because as Figure 17 shows, FixedRW(Rand) becomes better than Replicated-D and FixedR-D(Rand) when CV_i

becomes larger.

When Round Robin allocation is used instead, the results for unity CV_i shown in Figure 21 resembles that of Figure 16. At higher CV_i , FixedR-C(RR) is observed to degrade in performance relative to Replicated-C when F_i increased. The results are shown in Figure 22. As is the case with Random allocation, the FixedR-C and FixedRW policies appear to be more sensitive to F_i at higher values of CV_i . This may be because of the combined effect of a job spending more time doing I/O (larger F_i) and the presence of very large I/O requests (as a result of larger CV_i) making the policies more sensitive to poor workload distribution.

FIGURE 21. R vs F_1 for O1 workloads and Round Robin device allocation

FIGURE 22. R vs F_1 for O1 workloads, Round Robin allocation and higher CV_i .

5.4 Results for NOP workloads

Effects of Job Arrival Rate (λ):

Figure 23 shows the effect the data distribution policies have on the mean job response

times as a function of λ for NOP workloads and Random device allocation. FixedR-C(Rand) policy becomes noticeably inferior to Replicated-C at moderate to high λ , unlike the results for OI workloads. The FixedRW(Rand) policy appears to perform slightly poorer compared to the Replicated-D and FixedR-D(Rand) policies at high λ . This behaviour by the FixedRW(Rand) policies appears to be independent of the number of parallel I/O operations per NOP job because the same behaviour is seen even when n_i is increased from 5 to 20 (see Figure C-5 in Appendix C). The FixedR-D(Rand) is expected to be iden-

tical to Replicated-D because devices are selected randomly by both policies during the simulation.

Changing to Round Robin device allocation makes FixedRW(RR) perform better than FixedR-D(RR), which in turn performs better than Replicated-D policy. The change in allocation policy does not seem to alter the tendency of the FixedR-C(RR) policy to diverge rapidly from Replicated-C at moderate to high λ (Figure 24). The disparity in the

behaviour of the FixedR-C policies between O1 and NOP workloads suggests the absence of CPU-I/O overlap in NOP jobs contributes to the degradation of FixedR-C policies at moderate to high λ .

Effects of Coefficient of Variation of I/O Demand Per Request (CV_i):

The behaviour of the data distribution policies with Random allocation for NOP workloads and increasing CV_i (see Figure C-6 in Appendix C) is similar to that shown in Figure 23 for behaviour as a function of λ . FixedRW(Rand) policy for NOP workload is slightly inferior to Replicated-D. FixedR-C(Rand) becomes inferior to Replicated-C with increasing CV_i but is still superior to FixedRW(Rand) or Replicated-D.

The behaviour of the policies using Round Robin allocation are similar to those shown in Figure 24. The two policies with centralized management perform better than the decentralized policies. Of the centralized policies, Replicated-C performs better than FixedR-C(RR), particularly as CV_i increases. The results are shown in Figure C-7 in Appendix C.

Effects of Job I/O Factor (F_i):

At unity CV_i (Figure C-8 in Appendix C), the results resemble those in Figure 23. FixedR-C(Rand) diverges from Replicated-C starting at low F_i , but remain superior to the distributed policies and the FixedRW(Rand) policy. Changing to Round Robin allocation (Figure C-9 in Appendix C), the FixedR-C(RR) policy is comparable with Replicated-C at low to moderate F_i , but starts to diverge at higher F_i . The FixedRW(RR) policy continues to perform between that of FixedR-C(RR) and FixedR-D(RR).

At higher values of CV_i , FixedR-C(Rand) and FixedR-C(RR) diverge quickly from Replicated-C as F_i increases. The results are shown in Figure C-10 and Figure C-11 in Appendix C for Random and Round Robin allocation, respectively. These results resem-

ble those shown in Figure 23 and Figure 24 for Random and Round Robin allocation, respectively. FixedRW(Rand) performs slightly worse than Replicated-D while FixedRW(RR) performs better than Replicated-D. For both device allocation policies at higher values of CV_i , the difference in performance between FixedR-C and Replicated-C increases with F_i .

5.5 Discussion

The experimental results reveal insights into the effects of different data distribution policies on the mean response times of jobs. These insights are summarized below.

The experimental results suggest that where the cost of maintaining a centralized database of I/O device status and the cost of centralizing all I/O request routing operations are small, centralized data distribution policies consistently gives lower mean job response times than the decentralized policies. Of the two centralized policies studied, Replicated-C is seen to be a better choice than FixedR-C when the I/O demand of requests have large variations (CV_i) for job models with and without CPU-I/O overlap. The trade-off between these two policies is that Replicated-C requires replication of read data whereas FixedR-C uses no data replication at all. The superiority of Replicated-C over FixedR-C can be attributed to the poorer utilization of available of devices by jobs to handle FixedR-C read I/O requests because fixed device allocation can result in requests being sent to busy devices even though other devices are idle. The comparable performance of FixedR-C to Replicated-C under certain conditions, especially for OI jobs, suggests that the centralized, spreading of write I/O workload across the devices can have a stronger effect on job

response time than the fixed handling of reads.

The difference between centralizing and decentralizing the routing of I/O request can be rather large, assuming the cost of centralizing operations in itself is not prohibitively high. Although both Replicated-C and Replicated-D replicates read data, and any request can be serviced by any I/O device, Replicated-C is seen to result in consistently lower job response times than Replicated-D. This can be attributed to the central router being able to more evenly distribute the I/O workload across all the available I/O devices because the router holds requests until an idle device is available. Contrast this to Replicated-D router which, because it does not know the device statuses when it sends a request to an I/O device, causes the I/O workload to be unevenly distributed across the devices.

Where centralized routing of I/O requests is not feasible, a decentralized data distribution policy such as FixedRW with Round Robin device allocation can be a good choice. With the round robin distribution of data, and hence the I/O workload, across all available devices, the FixedRW(RR) policy is seen in the experiments to perform better than the two distributed policies for both NOP and OI jobs. This spreading of the I/O workload increases I/O parallelism and help reduce device bottlenecks and I/O wait times. In addition, this data distribution policy does not need data replication. Studies with FixedRW using Random device allocation strongly indicates that should data become poorly (ie. appear randomly) distributed, the data should be redistributed to preserve the round robin distribution so that a large degree of I/O parallelism can be maintained.

When FixedR-D policy is coupled with Random device allocation for the read I/O

operations, both the writes and reads are essentially sent to randomly chosen devices. FixedR-D(Rand) policy is therefore expected to behave essentially the same as Replicated-D. Using Round Robin allocation for the FixedR-D reads is seen to improve this policy's performance over the Replicated-D, suggesting the Round Robin distribution of the read operations compensates in part for the load imbalance caused by the poorer Random distribution of FixedR-D(RR) write workload.

Chapter 6 Job Based I/O Scheduling Policies

Overhead costs associated with doing disk I/O include the disk seek and rotational latency times, both of which are incurred while positioning the disk head over the physical location of the desired data. These disk overheads can have a major impact on I/O performance, particularly if the overheads are significant relative to the data transfer times. In response, disk scheduling algorithms such as SSTF, SCAN and LOOK [Silb92] were proposed to attempt to reduce the overhead costs by reducing the amount of disk head movement needed to service consecutive I/O requests. These disk scheduling algorithms remain an important part of modern disk I/O systems.

The five new job based I/O scheduling policies presented in this thesis complement the disk scheduling algorithms by providing a higher level of scheduling intended to handle applications with data transfer times that are much greater than the access overheads. Because the access overhead is assumed to be relatively small (for example, if data for a given application were stored contiguously on a disk such that only a single seek is required), scheduling is no longer based on minimizing the access overhead. Instead, the scheduling policies schedule requests based on the relative characteristics of the jobs that issue the requests.

Four of the five policies use job characteristics such as the age of the jobs ("Oldest Job First"), the total outstanding I/O demand of the jobs ("Smallest Total Outstanding I/O Demand" and "Largest Outstanding I/O Demand") or the number of concurrent jobs in the system ("Round Robin") to determine the highest priority job. The job with the highest

priority is the job from which an outstanding I/O request is serviced by an idle I/O device (see Figure 25a). Each job has a queue into which the job's I/O requests are put if the requests cannot be serviced immediately.

FIGURE 25. No job based scheduling versus job based scheduling

In addition to the above four job based scheduling policies, a policy called "JBNOT" is also introduced. This policy does not schedule I/O requests based on job priorities, but instead services all requests on a FCFS basis (i.e. no job based scheduling) (see Figure 25b). JBNOT is useful as a benchmark against which the performance of the other four policies are compared to see if any significant performance gains can be obtained by using job based I/O scheduling.

In job based I/O scheduling, an I/O scheduler uses one of the job based I/O scheduling policies first to determine the highest priority job, then to select an I/O request from that job. To simplify the experiments, requests from a job are processed by the I/O scheduler in the order they are issued by the job. A more sophisticated approach may be to apply something like a disk scheduling algorithm to select a request from within a job.

Some of the job based I/O scheduling policies have analogies with processor scheduling algorithms such as FCFS, Shortest Job First (SJF) and Round Robin (RR). The

JBNOT policy is analogous to the FCFS algorithm which gives the idle CPU to the process that requests it first. The JBSOIO policy is analogous to the SJF algorithm which gives the CPU to the process with the smallest next CPU burst. Note that the SJF processor scheduling algorithm is the optimal non-preemptive policy because it produces the minimum mean waiting time. 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 [Silb92]. The JBRR policy is analogous to the RR algorithm which gives the CPU to each process for a fixed time quantum before preempting the process to give the CPU to the next waiting process. An important distinction between the I/O scheduling policies and the CPU scheduling algorithms is that the latter can pre-empt processes while the former cannot pre-empt I/O requests, but must let each I/O request run to completion.

Job based I/O scheduling can be done either centrally by a central scheduler or by schedulers local to each I/O device. A central scheduler is appropriate if a centralized data distribution policy such as Replicated-C which performs centralized routing of I/O requests can be employed. Local schedulers are appropriate if a distributed or fixed data distribution policy such as Replicated-D, FixedR-D or FixedRW which performs distributed or fixed routing of I/O requests were used. A data distribution policy such as FixedR-C which does both centralized and fixed routing of requests would require central and local schedulers that operate independently of each other.

When an I/O request cannot be serviced immediately, it is put into an I/O queue. A centralized job based scheduler maintains a set of M I/O queues, one queue for each job in the

system. If the data distribution policy specifies that a read or write request is to be centrally managed, and no idle device is available to service the request, then that request is put into the I/O queue associated with the job that issued the request. For example, a request from job 'i' would be put into the queue ' Q_i ', the queue for job 'i', if no idle devices is available to handle the request. Each local scheduler maintains its own set of M I/O queues, one queue for each job in the system. If the data distribution policy specifies that a read or write request is to be locally managed (ie. decentralized management), and the device to which the request is sent is not idle, then the request is put into the I/O queue associated with the job that issued the request. For example, a request from job 'i' to device 'j' would be put into queue ' Q_{ji} ', the queue at device 'j' for job 'i', if the device is not idle. If JBNOT is the scheduling policy, each I/O queue set only has one I/O queue.

Of the jobs that have requests waiting in a scheduler's I/O queues, a scheduler (either central or local) determines the highest priority job and selects a request from the I/O queue associated with that job when an I/O device becomes idle. Each scheduler uses only information available from its set of M I/O queues to determine which job is highest priority. As a result, the highest priority job for one scheduler might not be the highest priority job for another scheduler at any given instant in time. A more detailed description of how each job based I/O scheduling policy determines the highest priority job is given below.

The *JBNOT* ("Not Job Based") policy does not assign priorities to jobs but instead selects I/O requests in the order they are received by the scheduler. For example, a central scheduler selects requests in the order they arrive in the central I/O queue while a local

scheduler selects requests in the order they arrive in its local I/O queue.

The *JBOJF* ("Job Based-Oldest Job First") policy gives higher priority to the job with the earlier job arrival time (the time when the job enters the system). A scheduler looks up the arrival time of each job which has waiting requests in its I/O queues and the job with the earliest arrival time becomes the highest priority job for that scheduler. I/O requests from older jobs are always selected before requests from younger jobs. Equivalently, a newly started job will have its I/O requests selected by the scheduler only if no older jobs have outstanding requests in the scheduler's I/O queues.

The *JBLOIO* ("Job Based-Largest Outstanding I/O") policy gives higher priority to the job with the larger total outstanding I/O demand. Outstanding I/O demand is defined as the demand of a request that has been issued but not yet serviced. Each scheduler uses only the requests that are waiting in its M I/O queues to calculate total outstanding I/O demand in each of its M queues. The queue with the largest value of total outstanding I/O demand is the queue from which the scheduler selects a request. In other words, the job associated with that queue is the highest priority job. Because total outstanding I/O demand as calculated by a scheduler varies over time, whenever a device becomes idle, job priorities must be re-evaluated by the scheduler associated with that I/O device.

The *JBSOIO* ("Job Based-Smallest Outstanding I/O") policy is like the *JBLOIO* except that higher priority is given to the job with the smaller total outstanding I/O demand. Like *JBLOIO*, the priority of each job must be re-evaluated by the scheduler whenever an associated I/O device becomes idle.

The *JBRR* ("Job Based-Round Robin") policy gives priority to jobs in a sequential manner. A scheduler looks at each of its M I/O queues in sequential order for an outstanding request whenever an associated I/O device becomes idle. If a job has a request waiting at the queue the scheduler is currently looking at, then that I/O request is serviced by the idle device, otherwise the scheduler looks in the next I/O queue for outstanding requests.

The I/O scheduling policies work closely in conjunction with the data distribution policies studied in the previous chapter. The distribution of data according to a given distribution policy determines how I/O requests, and hence the I/O workload, is distributed amongst the I/O devices. The requests are serviced by an idle I/O device in an order determined by the job based I/O scheduling policy currently in use.

To illustrate how these different policies are applied by a scheduler (either locally by a local scheduler or centrally by a central scheduler), consider an example where a system has four running jobs with job information as shown in Table 4. The source of the next request for the idle device associated with the scheduler comes from the job which has the '*' marked underneath it for the corresponding scheduling strategy being used. Note that for *JBRR*, job #4 had priority last time, so this time, job #1 has priority. The scheduler finds job #1 has no waiting I/O requests in the associated I/O queue, so the queue for the next job, job #2, is checked. Because job #2 has outstanding requests in its I/O queue, the scheduler selects a request from job #2 for servicing by the idle device.

TABLE 4. Example of applying job based I/O scheduling

Job Info & Scheduling Policy	Job #1	Job #2	Job #3	Job #4
Job Arrival Time	$t = t_0 + 3$	$t = t_0 + 6$	$t = t_0$	$t = t_0 + 9$
Amt of Outstanding I/O	0	10	5	2
Round Robin Check	check this job this time	-----	-----	checked last time
JBOIF policy			*	
JBLOIO policy		*		
JBSOIO policy				*
JBRR policy		*		

6.1 The Performance of I/O Scheduling Policies

The performances of the I/O scheduling policies are presented in this section. Only two job models are used in order to reduce the size of the parameter space for the experiments. OI and NOP job models are chosen to represent jobs with and without CPU-I/O overlap characteristics, respectively. This presence or absence of this overlap was found in Chapter 4 to affect a job's mean job response time.

Experiments are run using four data distribution policies: Replicated-C, Replicated-D, FixedRW(Rand) and FixedRW(RR). Experiments from Chapter 5 show that Replicated-C is a very good policy while Replicated-D and FixedRW(Rand) are often poor policies and FixedRW(RR) is a good compromise between Replicated-C and Replicated-D. Because Replicated-C and Replicated-D data distribution policies show extremes in performance, the results for these two policies are presented and discussed explicitly in this thesis. References to results for FixedRW(RR) and FixedRW(Rand) are made in the text when applicable. Some of the data for FixedRW policies have been included in the Appendix for completeness.

To keep the parameter space manageable and to maintain focus on I/O, a number of parameters are kept fixed through the experiments. A description of these parameters and their values is given in Chapter 5.

Three principal experiments are used to investigate the impact of job based I/O scheduling policies on the mean job response times: as a function of λ ; as a function of CV_i ; as a function of F_i . Increasing λ is used to increase system load and the probability that multiple jobs run concurrently. Increasing CV_i aids in understanding how the policies perform when the I/O demand per request varies markedly. Increasing F_i is used to increase the mean total I/O demand of a job.

To present the data from the many experiments, the results are organized and presented in the following manner:

- Replicated-C results for
 - O1 job results for
 - One I/O device
 - Several I/O devices
 - NOP job results for
 - One I/O device
 - Several I/O devices
- Replicated-D results for
 - O1 job results for several I/O devices
 - NOP job results for several I/O devices

6.2 Replicated-C Results for O1 Workloads and One I/O Device

Effects of Job Arrival Rate (λ):

When Replicated-C policy is used with only one shared I/O device, I/O contention between the jobs is managed to varying degrees of effectiveness by the I/O scheduling policies as shown in Figure 26 for increasing λ . With O1 workload, the JBSOIO and JBLOIO policies produce the best and worst results, respectively, of the five job based scheduling policies. The performance of the JBRR and JBNOT policies, both of which give similar results, lie between the JBSOIO and JBLOIO policies while the JBOJF policy is seen to be comparable to the JBSOIO policy.

The similarity in performance between the JBOJF and JBSOIO policies is believed to

be related to the I/O pattern of the O1 job model. The O1 model I/O pattern is characterized by a burst of read I/O operations when the job starts running, followed by intermittent write I/O operations from child threads that have finished doing computations. Intermittent write I/O requests are, therefore, characteristic of older O1 jobs. Because the write operations are intermittent, the total outstanding I/O demand for an older O1 job at any given device is likely to be small so the job that is selected by JBOJF policy is often also the same job selected by the JBSOIO policy. Consequently, the performance difference between servicing a JBSOIO job or JBOJF job tends to be small.

The striking similarity in performance between JBRR and JBNOT policy implies that the performance difference between processing I/O requests from O1 jobs in a round robin manner and processing them in the order they arrive at the device can be small when averaged over a long period of time. This behaviour by these two policies may be because of the intermittent nature of O1 write requests. Because they are intermittent, only a small number of these requests from a job is expected at the device at any given time so that on average, servicing them in a round robin fashion or in the order they arrive at the device makes little difference. This suggests perhaps the intermittent nature of the write I/O requests from older O1 jobs has a major influence on the relative performances between these two policies.

Effects of Coefficient of Variation of I/O Demand Per Request (CV_i):

Increasing CV_i dramatically alters the performance of the JBOJF policy relative to the JBSOIO policy (see Figure 27). The JBOJF policy rapidly diverges from the JBSOIO pol-

icy as CV_i increases to become much inferior to JBSOIO at high CV_i . JBSOIO remains the best policy, especially when CV_i becomes large. The JBRR and JBNOT policies do not diverge from each other, but instead maintain comparable performance throughout the same range of CV_i . This reinforces the suggestion that the nature of intermittent write I/O requests from older O1 jobs have a major impact on the relative performances between these two policies.

The growing difference between JBOJF and JBSOIO policies as CV_i increases can be tied back to the I/O pattern of the O1 job model. When CV_i is low, the job chosen by JBOJF and the job chosen by JBSOIO is often the same job because an older O1 job (the job chosen by JBOJF) sends write requests intermittently so that it also tends to be the job

with the smallest total outstanding I/O demand. As CV_i increases, the variation in I/O demand of each I/O request also increases which means the I/O demands of the intermittent write requests from the older O1 job chosen by JBOJF can be rather large. This reduces the likelihood that both JBOJF and JBSOIO choose the same job, leading to a growing difference in performance between the two policies.

Effects of Job I/O Factor (F_i):

Increases in F_i at unity CV_i produce relative scheduling policy performances similar to that obtained by increasing λ (see Figure 26). When CV_i is larger, however, increasing F_i brings about a rapid degradation in JBOJF performance relative to JBSOIO policy (see Figure 28). These results seem to indicate that the JBOJF policy performance is more sensitive to increases in F_i when CV_i is relatively large. The JBRR policy is again seen to perform comparably with JBNOT for increasing F_i .

6.3 Replicated-C Results for O1 Workloads and Several I/O Devices

Effects of Job Arrival Rate (λ):

Increasing the number of I/O devices should reduce the I/O device contention problem so that the choice of I/O scheduling policy should become less important. As the number of I/O devices is increased (eg. from $I=1$ to $I=2$), the relative rankings of the scheduling policies are seen to be the same as for $I=1$, but the differences between policies become smaller. At $I=3$ and $I=4$, the results show little advantage in using job based I/O scheduling over no scheduling (ie. JBNOT) even as the system approaches saturation (high λ). The results for O1 workload with $I=4$ I/O devices are shown in Figure 29.

FIGURE 28. R vs F_i for O1 workloads, Replicated-C, 1 I/O device

Effects of Coefficient of Variation of I/O Demand Per Request (CV_i):

As the number of I/O devices increases, the relative behaviours of the job based scheduling policies with respect to increases in CV_i remain the same as for $I=1$ but the differences between policies decrease. As the first few devices are added, the JBOJF policy performance can still be seen to degrade relative to JBSOIO policy as CV_i increases, while the JBRR and JBNOT policies still remain comparable to each other. When I is sufficiently large (Figure 30), the relative differences between the five policies become very small over a wide range of CV_i . This would imply that the system has enough I/O devices to use with centralized input data replication and centralized routing to alleviate enough contention for I/O devices so that a simple I/O scheduling policy such as JBNOT is adequate over a wide range of CV_i values.

Effects of I/O Factor (F_i):

Figure 31 shows how the job based scheduling policies behave when F_i increases for unity CV_i . Like the case for $I=1$ I/O devices, the same patterns of relative scheduling performances are seen when $I=4$. The JBOJF policy and JBSOIO policy have comparable performances as do the JBRR and JBNOT policies. The relative differences in mean job response times between the policies do not increase greatly with increases in F_i . At larger CV_i , the difference between the JBOJF and JBSOIO increases while the differences between the JBRR, JBNOT and JBOJF policies decrease as a function of F_i , but these changes are relatively minute when the system has enough of I/O devices. The results for larger CV_i are shown in Figure 32.

6.4 Replicated-C Results for NOP Workloads and One I/O Device

Effects of Job Arrival Rate (λ):

When a NOP workload is used, the relative rankings of the JBLOIO, JBRR and JBSOIO policies remain unchanged from that for the O1 workload. The JBSOIO policy produces the best results while the JBLOIO policy produces the worst results. The JBRR policy performs in between that of JBSOIO and JBLOIO policies. The primary difference between the NOP and O1 results is the relative performances of the JBNOT and JBOJF policies. The JBOJF policy is no longer very close to the JBSOIO policy with NOP workloads, while the JBNOT policy has lower mean job response times than JBRR. This is shown in Figure 33. JBNOT and JBOJF policies perform almost identically.

Comparing the NOP and O1 results for a single I/O device reveals for both types of workloads, the JBLOIO policy produces the largest mean job response times and the JBRR produces the second largest response times. JBSOIO policy has thus far shown it gives the lowest mean job response times, but a policy such as JBOJF for O1 workload are able to approach the performance of JBSOIO.

The JBLOIO and JBRR policies' results point to the possibility that the performance of these two job based I/O scheduling policies have little or no dependency on the I/O patterns of the applications. One explanation for the JBLOIO and JBRR observations is given in terms of "job switching" by I/O devices as a result of the scheduling policy being used by the I/O scheduler. Job switching refers to the switching of the job whose I/O requests

2 OF 2

PM-1 3 1/2" x 4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

FIGURE 32. R vs F_i for O1 workloads, Replicated-C, 4 I/O devices at higher CV_i .

are being serviced by an I/O device, from job 'i' to another job 'k', even though some requests from job 'i' are still queued waiting for service. If after having selected a job as the high priority job, the scheduler determines later that a different job has higher priority, then the device is forced to "switch jobs" and begin servicing requests from the higher priority job. This job switching can increase the time a request must wait for service from the I/O device and therefore increase the response time of a job.

Both the JBLOIO and JBRR policies have a tendency to cause frequent job switching. A job selected by the JBLOIO policy at a given device has the largest total outstanding I/O demand at that device. After a few of this job's requests have been serviced the device, the likelihood that this job will not remain the highest priority job is quite high because its

total outstanding I/O demand has decreased. At this point, the device is susceptible to start servicing existing requests from a different job with a larger total outstanding I/O demand at that device. A NOP job, because it always sends n_i request during each I/O burst, can easily cause a device to switch jobs if the device receives several of these new requests from the job and the total I/O demand of the newly received requests is larger than the sum of the existing I/O demand from other jobs at that device.

The JBRR policy inherently job switches because of the round robin nature of the job selection process. The more jobs that have outstanding I/O requests at a given device, the more waiting a request is likely to do before receiving service. Because the job switching done by JBRR policy is systematic and has no potential to cause a job's requests to starve

obtained using Replicated-C for NOP workload with a single I/O device (Figure 33). The JBLOIO policy gives the largest mean job response times while the JBRR policy gives the second largest times and the JBSOIO is the policy which gives the best times. The JBNOT and JBOJF policies produce mean job response times between those for JBSOIO and JBRR policies but unlike Figure 33, JBNOT is seen to be superior to the JBOJF policy.

This superiority of JBNOT over JBOJF was seen for NOP workloads with Replicated-C policy when I/O devices were added to the system to reduce contention for the shared devices. When centralized routing is employed to route requests to a single I/O device, that device must schedule n_i requests from a NOP job whenever that NOP job does I/O. By adding more devices, the average number serviced by a device for each job becomes less than n_i . With distributed (or fixed) routing, any given I/O device is not likely to receive all n_i requests from a NOP job. Instead, a device is more likely to receive only some of the n_i requests sent by the NOP. Consequently for NOP workloads and non-central routing, the performances of the job based I/O scheduling policies with many I/O devices (eg $I=4$) is more akin to that of centralized routing with a few I/O devices, rather than a single I/O device (eg. $I=2$). FixedRW(Rand) and FixedRW(RR) results show similar trends. Figure C-17 in Appendix C shows the results for FixedRW(RR).

This is in contrast to the findings for O1 workloads where the decentralized handling results with several devices bear striking resemblance to results for centralized handling with one I/O device. I/O requests from O1 jobs are spread out over time rather than being bursty so that a device is likely to receive only a few requests at a time from different jobs,

NOP I/O workloads.

The JBNOT policy is vulnerable to job switching if requests are sent intermittently over time (like the O1 write requests) rather than in bursts (like the NOP read/write or O1 read requests) because this policy services requests in the order they are received by the device. The reason is that with intermittent requests, a request in the single I/O queue is likely to be followed by a request from a different job while with bursty requests, several requests from the same job are likely to follow each other. When handling O1 read bursts, JBNOT policy results in less job switching than the JBRR but when handling the intermittent O1 write requests, JBNOT policy results in job switching comparable to JBRR. Consequently, JBNOT should perform better than or at least comparable with the JBRR policy for O1 workloads.

For NOP workloads, n_i requests are sent at the same time during an I/O burst so a device will always complete all the requests it receives from one NOP job before starting on the requests from another NOP job. Compared to JBOJF policy, JBNOT policy causes less job switching with NOP jobs so it performs better than or at least no worse than JBOJF policy.

Effects of Coefficient of Variation of I/O Demand Per Request (CV_i):

The relative performances of the policies as CV_i increases remain the same as that shown in Figure 33, with JBSOIO being the clear favourite especially at high CV_i (Figure 34). No clear winner is evident between the JBNOT and JBOJF policies.

Effects of Job I/O Factor (F_i):

The results obtained by increasing F_i at unity CV_i show relative performances similar to those in Figure 33. The JBSOIO policy gives the best results and the difference between this policy and the JBOJF and JBNOT policies do not increase dramatically as F_i increases. At higher values of CV_i , the difference between JBSOIO and the JBNOT and JBOJF policies become larger as F_i increases (see Figure 35). The performances for JBNOT and JBOJF policies appear identical.

6.5 Replicated-C Results for NOP Workloads and Several I/O Device

Effects of Job Arrival Rate (λ):

As the number of I/O devices increases, the mean job response times of the JBNOT

policy is seen to approach those of the JBSOIO policy. Meanwhile, the mean job response times for the JBOJF policy approach those of the JBRR policy. This trend in performances by the JBNOT and JBOJF policies is most noticeable as the first few devices are added to the system (see Figure C-12 for $I=2$ results in Appendix C). When the system has sufficient number of I/O devices, such as when $I=4$, the differences in performances between the five policies become small but this performance improvement by JBNOT over JBOJF is still visible as shown in Figure 36.

Effects of Coefficient of Variation of I/O Demand Per Request (CV_i):

The improvement in JBNOT performance over JBOJF policy as CV_i increases also occurs as more I/O devices are added to the system (see Figure C-13 for $I=2$ results in

Appendix C). With four devices, all the policies seem to perform comparably so that the JBNOT policy appears adequate as shown in Figure 37.

Effects of Job I/O Factor (F_i):

As I/O devices are added, the JBOJF policy performance as a function of increasing F_i also approaches that of JBRR while JBNOT approaches that of JBSOIO. At unity CV_i with four I/O devices, Figure 38 shows the mean job response times of JBRR and JBOJF policies appear indistinguishable and that JBNOT policy compares well with JBSOIO. At larger values of CV_i , the same experiment shows performance differences between the five policies are smaller and performance gains from using job based scheduling is less evident until high F_i where JBSOIO policy is seen to be the better policy (see Figure 39).

6.6 Replicated-D Results for O1 Workloads with Several I/O Devices

Effects of Job Arrival Rate (λ):

Figure 40 shows results obtained with O1 jobs for increasing λ are very similar to those obtained using Replicated-C with one I/O device. The JBOJF policy produce comparable results to the JBSOIO policy while the JBRR policy produce almost identical results to the JBNOT policy. Both JBSOIO and JBOJF policies are better than no job based I/O scheduling (JBNOT). Experiments using FixedRW(RR) and FixedRW(Rand) data distribution policies shows similar performances by the scheduling policies. A plot of the FixedRW(RR) results appear in Figure C-14 in Appendix C.

The striking similarity between the results in Figure 40 for several devices and the

FIGURE 38. R vs F_i for NOP workloads, Replicated-C, 4 I/O devices, low CV_i .

$M=4, P=20, I=4, d_s=0.01, CV_s=1.0, n_d=10, d=1.0, CV_d=1.0, n_i=5, CV_i=1.0, \lambda=0.4.$

FIGURE 39. R vs F_i for NOP workloads, Replicated-C, 4 I/O devices, higher CV_i .

$M=4, P=20, I=4, d_s=0.01, CV_s=1.0, n_d=10, d=1.0, CV_d=1.0, n_i=5, CV_i=3.0, \lambda=0.4.$

results in Figure 26 for a single I/O device is believed to be primarily a consequence of the distributed handling of I/O requests by I/O schedulers local to each device. When non-centralized routing of I/O requests is performed, as is the case for Replicated-D or the FixedRW policies, the requests are automatically sent to an I/O device and the requests at each device are scheduled by a local I/O scheduler. Although a system as a whole may have more than one I/O device, the local scheduler at each device has access to only a single I/O device to which it can schedule I/O requests from multiple jobs.

Effects of Coefficient of Variation of I/O Demand Per Request (CV_i):

Fixing λ and increasing CV_i (see Figure 41) shows trends consistent with those seen using Replicated-C with one I/O device (see Figure 27). The mean job response times using JBOJF policy become increasingly greater than those of JBSOIO as CV_i increases.

making JBOJF inferior to JBSOIO. JBRR policy performs equally well compared to JBNOT policy, even when CV_1 is high. JBLOIO policy is still the worst policy while JBSOIO is the best policy. Similar performance behaviours are observed when using FixedRW(RR) and FixedRW(Rand) data distribution policies instead of Replicated-D

Results for FixedRW(RR) are shown in Figure C-15 in Appendix C.

Effects of Job I/O Factor (F_1):

The rankings of the policies as F_1 increases for unity CV_1 are similar to that shown in Figure 26. JBSOIO performs better than JBOJF, especially at high F_1 . The JBRR and JBNOT policies are seen to be comparable to each other.

At higher values of CV_i , JBOJF policy causes the mean job response times to become greater than those of JBSOIO at moderate to high F_i . The JBRR and JBNOT both continue to give similar mean job response times. The graph of the results are shown in Figure 42.

The difference in behaviour by the JBOJF policy as a function of F_i at low and higher CV_i values seems to suggest this policy is more sensitive to increases in F_i when CV_i is large than when CV_i is small. This observation for JBOJF policy is consistent with the results obtained using Replicated-C with one I/O device, which further reinforces the belief that non-centralized routing with several I/O devices is related to centralized routing with a single I/O device. Substituting Replicated-D with FixedRW(RR) or FixedRW(Rand) produces similar behaviours by the scheduling policies. FixedRW(RR)

results are shown in Figure C-16 in Appendix C for larger CV.

6.7 Replicated-D Results for NOP Workloads with Several I/O Devices

With only a single I/O device, the entire I/O workload is always sent to that device regardless of the data distribution policy in effect. The results from using Replicated-D, for example, is expected to be the same as for Replicated-C. Consequently, no experiments are performed using another policy such as Replicated-D with a single I/O device.

Effects of Job Arrival Rate (λ):

Replicated-D policy results using NOP workload in a system with several I/O devices is shown in Figure 43 for increasing λ . These results bear some resemblance to results

obtained using Replicated-C for NOP workload with a single I/O device (Figure 33). The JBLOIO policy gives the largest mean job response times while the JBRR policy gives the second largest times and the JBSOIO is the policy which gives the best times. The JBNOT and JBOJF policies produce mean job response times between those for JBSOIO and JBRR policies but unlike Figure 33, JBNOT is seen to be superior to the JBOJF policy.

This superiority of JBNOT over JBOJF was seen for NOP workloads with Replicated-C policy when I/O devices were added to the system to reduce contention for the shared devices. When centralized routing is employed to route requests to a single I/O device, that device must schedule n_i requests from a NOP job whenever that NOP job does I/O. By adding more devices, the average number serviced by a device for each job becomes less than n_i . With distributed (or fixed) routing, any given I/O device is not likely to receive all n_i requests from a NOP job. Instead, a device is more likely to receive only some of the n_i requests sent by the NOP. Consequently for NOP workloads and non-central routing, the performances of the job based I/O scheduling policies with many I/O devices (eg $I=4$) is more akin to that of centralized routing with a few I/O devices, rather than a single I/O device (eg. $I=2$). FixedRW(Rand) and FixedRW(RR) results show similar trends. Figure C-17 in Appendix C shows the results for FixedRW(RR).

This is in contrast to the findings for OI workloads where the decentralized handling results with several devices bear striking resemblance to results for centralized handling with one I/O device. I/O requests from OI jobs are spread out over time rather than being bursty so that a device is likely to receive only a few requests at a time from different jobs,

regardless of whether the centralized or decentralized management is used.

Effects of Coefficient of Variation of I/O Demand Per Request (CV_i):

Figure 44 illustrates the results of increasing CV_i for NOP workloads. The relative rankings of the policies are consistent with the results shown in Figure 43 and with the results for Replicated-C with a small number of devices. The data shows JBSOIO is the better policy because the mean job response times for JBNOT policy, the policy giving the second best results, increase faster than that for JBSOIO as CV_i becomes larger. FixedRW

policies' results show similar behaviours. FixedRW(RR) results are shown in Figure C-18 in Appendix C.

Effects of Job I/O Factor (F_i):

Figure 45 and Figure 46 shows how the job based I/O scheduling policies are affected

by increases in F_i at unity and higher values of CV_i , respectively. Both graphs show the relative rankings of the policies are consistent with that shown in Figure 43 and Figure 44. At higher CV_i , the JBRR and JBOJF policy performances approach that of JNOT. Both graphs also show that JBSOIO is the better policy at moderate to high F_i . Results using FixedRW(RR) and FixedRW(Rand) policies show similar behaviours. A plot of FixedRW(RR) results is given in Figure C-19 in Appendix C.

Effects of increasing parallel I/O operations (n_i):

Increasing the number of parallel I/O operations for NOP jobs, n_i , helps reduce the

mean job response times for all the job based I/O scheduling policies. Interestingly, as n_i increases, the performance of the JBNOT policy begins to approach that of the JBSOIO policy such that at a sufficiently large n_i (for example, $n_i=20$), the JBNOT policy produces lower mean job response times than the JBSOIO policy (see Figure 47). A graph showing similar results obtained with FixedRW(RR) is shown in Figure C-20 in Appendix C.

Effects of increasing n_i and CV_i :

By increasing CV_i , a CV_i value is eventually reached where the JBSOIO policy is once again superior to the JBNOT policy (see Figure 48). This CV_i value is found to be smaller for Replicated-D than for FixedRW(RR). The relative rankings of the JBOJF, JBRR and JBLOIO policies remain unaffected by increasing n_i .

In contrast, the JBSOIO policy usually performs the best out of the five job based I/O scheduling policies. One possible explanation for the performances stems from the idea that an I/O device continually processing waiting requests from different jobs introduces longer delays to waiting requests than if the device services waiting requests from the same job first.

Job switching can also account for the observation that the JBNOT and JBOJF policies have performances that lie between that of the JBRR and JBSOIO policies. The behaviour of JBNOT and JBOJF with respect to each other, however, can depend on the characteristics of the workload. For OI workloads, JBOJF policy is better than JBNOT while for NOP workloads, JBNOT is often better than JBOJF because of smaller number of job

that an I/O device continually processing waiting requests from different jobs introduces longer delays to waiting requests than if the device services waiting requests from the same job first.

Job switching can also account for the observation that the JBNOT and JBOJF policies have performances that lie between that of the JBRR and JBSOIO policies. The behaviour of JBNOT and JBOJF with respect to each other, however, can depend on the characteristics of the workload. For OI workloads, JBOJF policy is better than JBNOT while for NOP workloads, JBNOT is often better than JBOJF because of smaller number of job switches.

Another workload characteristic which is a factor is the number of parallel I/O opera-

tions in a NOP job. Increasing n_i can decrease the mean job response times of the JBNOT policy to values less than JBSOIO policy for lower values of CV_i . Experiments characterizing NOP models, however, show that increasing n_i also increases the access overhead component of each I/O request that may lead to an increase in job response times. This may limit how large n_i can become.

The choice of data distribution policy is a factor in determining whether job based I/O scheduling brings any benefits. For a sufficient number of I/O devices in the system, using Replicated-C policy decreases the mean job response time differences between the various scheduling policies such that JBNOT may be an adequate policy. Using a decentralized policy such as with Replicated-D or FixedRW(RR) with the same number of devices as Replicated-C, the differences between the policies are more pronounced such that job based I/O scheduling can be beneficial.

Experiments with the distributed and fixed routing of requests show job based I/O scheduling policy behaviour similar to that obtained with centralized routing (Replicated-C) using one or two I/O devices.

Chapter 7 Conclusions

7.1 Research Summary

This thesis describes a simulation based investigation of the impacts of different I/O models, data distribution policies and I/O scheduling policies on the performance of multiprogrammed parallel systems which support parallel I/O. Currently, little work which investigates the complex interactions taking place between I/O characteristics of workloads, the distribution of data and the scheduling of I/O requests exists. The results presented in this thesis indicate that a careful selection of policies and workloads can help to improve system performance. A paper presenting the results of the investigation between I/O applications and data distribution policies is to be presented at the Third International Conference of the Austrian Centre for Parallel Computing [Kwon96].

For a given number of I/O devices, significant improvement in mean job response times can be obtained by running jobs which, in general, do I/O in parallel, and which, specifically, have CPU-I/O overlap. A job which does I/O in parallel but does not overlap its I/O with computation can take longer to complete than a job which does I/O in parallel and has CPU-I/O overlap. Increasing the number of parallel I/O operations for jobs which do not have CPU-I/O overlap such that the number of parallel I/O operations (n_i) is greater than the number of child threads (n_d), coupled with an increase in I/O factor, can reduce their mean job response times below those of jobs with CPU-I/O overlap in both a uni- and multiprogrammed environment.

The presence of input data replication does not guarantee good performance as seen by

the performance disparity between Replicated-C and Replicated-D. Although both these policies permit any device to service any read or write request, Replicated-C performs consistently better than Replicated-D. Policies such as FixedR-C and FixedRW(RR) which have no input data replication also perform better than Replicated-D.

Centralized management and Round Robin device allocation have a bigger impact on system performance than input data replication. Replicated-C manages both read and write requests centrally and has the best performance of the data distribution policies studied. FixedR-C which manages write requests centrally demonstrates a performance often comparable to Replicated-C for O1 workload, and a performance better than the Replicated-D, FixedR-D and FixedRW policies for NOP and O1 workload.

On systems where I/O request management cannot be centralized, strong motivation exists to use fixed routing of requests together with Round Robin device allocation to spread the I/O workload across a large number of devices. FixedRW(RR) policy, for example, performs better than or comparably with the distributed policies for both NOP and O1 workload. Implicit in this is that little motivation exists for input data replication if a decentralized management approach is taken. Round Robin allocation can also improve the performance of policies such as FixedR-D which combine fixed and distributed routing. The better distribution of FixedR-D(RR) read I/O workload attained by using Round Robin allocation can compensate for the poor random routing of this policy's write requests.

The difference in system performance with and without job based I/O scheduling can

be very different on a system with a finite number of I/O devices depending on whether centralized or decentralized management is in effect. A system with four I/O devices, for example, may show JBNOT policy is adequate for both NOP and OI workloads if a centralized policy such as Replicated-C policy is in effect but may show job based I/O scheduling is beneficial if a decentralized policy such as Replicated-D is in effect. Replicated-C is able to distribute the I/O workload among the available devices such that job based I/O scheduling brings little benefit. In contrast, policies like Replicated-D or FixedRW(RR) are unable to distribute the I/O workload as well so job based I/O scheduling can be better than no job based scheduling. The experimental results show the performances of the job based scheduling policies obtained using decentralized management on systems with a larger number of I/O devices (eg. with four devices) resemble those obtained using centralized management on fewer devices (eg. one or two devices).

Where job based I/O scheduling is beneficial, the results show that NOP and OI workloads are often best scheduled using JBSOIO. With NOP jobs, increasing the number of parallel I/O operations per job such that the number of parallel I/O operations is greater than the number of child threads can sometimes make no job based I/O scheduling better than scheduling with JBSOIO.

7.2 Contributions

A number of contributions have been made in this thesis to address the I/O bottleneck issues. The NOP job model, a model whereby a job performs computation or I/O in distinct and non-overlapping stages is introduced in this thesis. The set of existing data distri-

bution policies is expanded by the addition of the FixedR class. In addition, the Round Robin device allocation scheme is integrated in the study of data distribution policies. Four new I/O scheduling policies which focus on scheduling I/O requests with large I/O demands are presented.

7.3 Suggestions for Future Research

Further work can be done with other job models which have parallel I/O, such as the O2 model, to characterize their performance with various data distribution policies and job based I/O scheduling policies. The O2 model can vary the degree of CPU-I/O overlap so that it may be useful to represent jobs which perform large number of I/O operations, each with a small I/O burst.

Further investigation of the job based I/O scheduling policies using the FixedR class of data distribution policy remains to be done. The investigation would study the effects of mixing fixed read request routing with centralized or distributed write request routing on I/O scheduling policy performance.

This study queues and services I/O requests within each I/O queue on a FCFS basis. Instead of using FCFS, a different queueing discipline such as smallest request first can be used. Alternatively, disk scheduling algorithms such as SCAN or SSTF can be incorporated with job based scheduling to provide a two tier scheduling strategy: one tier schedules using job related information while the next tier schedules using disk head location information.

Costs and overheads of scheduling, data replication and I/O request dispatching have been ignored in this study to permit concentration on higher level issues. Communication costs (eg. messaging costs) may be a concern for certain distributed systems which communicate over a communications network.

Each I/O device itself can be substituted by a group of devices, thus providing a multi-tiered data distribution scheme: data can be distributed across groups using one scheme, and then further distributed across the devices within the group using another scheme. This research assumes I/O devices are operated asynchronously. Synchronous operation of the I/O devices within a group of devices to simulate synchronized disk arrays may provide additional improvement in system performance.

All the experiments are done with jobs arriving as a Poisson process. Instead of arriving as a Poisson process, the jobs can be made to arrive in clumps. The job mix can also be changed from being a homogenous mix to a heterogeneous mix, which would more accurately reflect certain general purpose multiprogrammed systems.

This study has explored I/O management using simulations. Implementation of I/O management strategies on real systems is required to verify the promising results obtained through this exploratory study. Investigating the impact of buffering and channel congestion on I/O management is also important.

References

- [Bank84] J. Banks, J.S. Carson II, "Discrete-Event System Simulation", Prentice-Hall, New Jersey, 1984.
- [Chen93] S. Chen, D. Towsley, "The Design and Evaluation of RAID 5 and Parity Striping Disk Array Architectures", *Journal of Parallel and Distributed Computing*, vol. 17, 1993, pp. 58-74.
- [Chen95] S. Chen, D. Towsley, "A Performance Evaluation of RAID Architectures", *Journal of Parallel & Distributed Computing*, 1995.
- [Corb93] P.F. Corbett, D.G. Feitelson, J.-P. Prost, S.J. Baylor, "Parallel Access to Files in the Vesta File System", *Proc. Supercomputing SIGARCH 1993*, pp. 472-481.
- [Cyph93] R. Cypher, P. Messina, "Architectural Requirements of Parallel Scientific Applications with Explicit Communication", *Proc. International Symp. on Computer Architecture*, 1993.
- [Delr93] J.M. Del Rosario, R. Bordawekar, A. Choudhary, "Improved Parallel I/O via a Two-Phase Run-Time Access Strategy", *ACM Computer Architecture News*, Vol. 21, No. 5, December 1993, pp. 31-38.
- [Delr94] J.M. Del Rosario, A. Choudhary, "High Performance I/O for Massively Parallel Computers", *IEEE Computer*, March 1994, pp. 59-68.
- [Duss90] K. Dussa, B. Carlson, L. Dowdy, K.-H. Park, "Dynamic Partitioning in a Transputer Environment", *Proc. 1990 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, Boulder (CO), May 1990, pp. 203-213.
- [Feit95] D.G. Feitelson, P.F. Corbett, S.J. Baylor, Y. Hsu, "Parallel I/O Subsystem in Massively Parallel Supercomputers", *IEEE Parallel & Distributed Technology*, Fall 1995, pp. 33-47.
- [Gemm95] D.J. Gemmell et al, "Multimedia Storage Servers: A Tutorial", *IEEE Computer*, May 1995, pp. 40-49

- [Ghan93] S. Ghandeharizadeh, L. Ramos, "Continuous Retrieval of Multimedia Data Using Parallelism", IEEE Trans on Knowledge and Data Engineering, vol. 5, no. 4, August 1993, pp. 658-669.
- [Ghos90] D.Ghosal, S.K.Tripathi, G.Serazzi, S.H.Noh, P.Lenzi and A.K.Agrawala, "Characterizing Parallel Program Behavior: an Experimental Study", Institute for Advanced Computer Studies, University of Maryland, June 1990.
- [Jada95] D. Jadav, C. Srinila, A. Choudhary, P.B. Berra, "Techniques for Scheduling I/O in a High Performance Multimedia-on-Demand Server", Journal of Parallel and Distributed Computing, vol. 30, 1995, pp. 190-203.
- [Jain93] R. Jain, K. Somalwar, J. Werth, J.C. Browne, "Scheduling Parallel I/O Operations", ACM Computer Architecture News, vol. 21, no. 5, December 1993, pp. 47-54.
- [Jeon95] B-S. Jeong, E. Omiecinski, "Inverted File Partitioning Schemes in Multiple Disk Systems", IEEE Trans on Parallel and Distributed Systems, vol. 6, no. 2, February 1995, pp. 142-153.
- [Katz89] R.H. Katz, G.A. Gibson, D.A. Patterson, "Disk System Architectures for High Performance Computing", Proc of the IEEE, vol. 77, no. 12, December 1989, pp. 1842-1989.
- [Kim86] M.Y. Kim, "Synchronized Disk Interleaving", IEEE Trans on Computers, vol. 35, November 1986, pp. 978-988.
- [Koba81] H. Kobayashi, "Modeling and Analysis: An Introduction to System Performance Evaluation Methodology", Addison-Wesley, 1981.
- [Kotz94] D. Kotz, N. Nieuwejaar, "Dynamic File Access Characteristics of a Production Parallel Scientific Workload", Proc of Supercomputing '94, November 1994, pp. 640-649.
- [Krie94] O. Krieger, M. Stumm, "The Alloc Stream Facility - A Redesign of Application-Level Stream I/O", IEEE. Computer, March 1994, pp. 75-82.

- [Kwon96] P. Kwong, S. Majumdar, "Study of Data Distribution Strategies for Parallel I/O Management", Proc of the 3rd International Conference of the ACPC (Austrian Centre for Parallel Computing), Klagenfurt, Austria, 1996.
- [Lee93] E.K. Lee, R.H. Katz, "The Performance of Parity Placements in Disk Arrays", IEEE Trans on Computers, vol. 42, no. 6, June 1993, pp. 651-664.
- [Leut90] S. Leutenegger and M. Vernon, "The Performance of Multiprogrammed Multiprocessor Scheduling Policies", Proc. 1990 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, Colorado, May 1990, pp. 226-236.
- [Livn87] M. Livny, S. Khoshafian, "Multi-Disk Management Algorithms", Proc. 1987 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, May 1987, pp. 69-77.
- [Loug93] P. Lougher, D. Shepherd, "The Design of a Storage Server for Continuous Media", The Computer Journal, vol. 367, no. 1, 1993, p. 32-42.
- [McCa95] C. McCann, J. Zahorjan, "Scheduling Memory Constrained Jobs in Distributed Memory Parallel Computers", Proc. IFIP Performance '95- ACM SIGMETRICS '95, Joint International Conf. on Measurement and Modeling of Computer Systems, Ottawa (ON), pp. 208-219, May 1995.
- [Maju88] S.Majumdar, D.L.Eager, and R.B.Bunt, "Scheduling in Multiprogrammed Parallel Systems", Proc. 1988 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, Santa Fe (NM), May 1988, pp.104-113.
- [Maju91] S.Majumdar, D.L.Eager and R.B.Bunt, "Characterization of Programs for Scheduling in Multiprogrammed Parallel Systems", Performance Evaluation, Vol. 13 (1991), pp. 109-130.
- [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, 1994, pp. 298-307.

- [Maju95] S. Majumdar, F. Shad, "Characterization and Management of I/O on Multiprogrammed Parallel Systems", Proc. Seventh IEEE symposium in Parallel & Distributed Processing, San Antonio, Oct. 1995.
- [Niel95] J. Nielson, PARASOL 2.3 Users Guide and Tutorial. Dept of Systems & Computer Engineering, Carleton University, Ottawa, Canada, 1995.
- [Nieu95] N. Nieuwejaar, D. Kotz et al., "File-Access Characteristics of Parallel Scientific Workloads", Tech Report PCS-TR95-263, Dept. of Math. and Comp. Science, Dartmouth College, Hanover, U.S.A, August 1995.
- [Pasq94a] B.K. Pasquale, G.C. Polyzos, "Dynamic I/O Characterization of I/O Intensive Scientific Applications", Proc of Supercomputing '94, November 1994, pp. 660-669.
- [Pasq94b] J. Pasquale, E. Anderson, "Container Shipping - Operating System Support for I/O-Intensive Applications", IEEE Computer, March 1994, pp. 84-93.
- [Patt88] D.A. Patterson, G. Gibson, R.H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", Proc. 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.
- [Peri94] V.G.J. Peris, M.S. Squillante, V.K. Naik, "Analysis of the Impact of Memory in Distributed Parallel Processing Systems", Proc. 1993 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, Nashville (TE), May 1994 pp. 5-18.
- [Pres93] I.S. Pressman, S.A. Cook, J. Pacht, "The Optimal Placement of Replicas in a Network Using a READ ANY - WRITE ALL Policy", Proc CASCON '92 Conference, vol. 2, IBM Canada Ltd., Center for Advanced Studies, Toronto, November, 1992, pp. 189-201.
- [Pura95] A. Purakayastha et al, "Characterizing Parallel File-access Patterns on a Large-scale Multiprocessor", Int'l Parallel Processing Symposium, 1995, pp. 165-172.

- [Redd89] A.L.N. Reddy, P. Banerjee, "An Evaluation of Multiple-Disk I/O Systems", IEEE Trans. on Computers, Vol. 38, No. 12, December 1989, pp. 1680-1690.
- [Redd94] A.L.N. Reddy, J.C. Wyllie, "I/O Issues in a Multimedia System", IEEE Computer, March 1994, pp. 69-74.
- [Sale86] K. Salem, H. Garcia-Molina, "Disk Striping", IEEE International Conf on Data Engineering, 1986, pp.336-342.
- [Seti93] S.K. Setia, M.S. Squillante, S.K. Tripathi, "Processor Scheduling on Distributed Memory Parallel Computers", Proc. 1993 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, Santa Clara (CA), May 1993 pp.158-170.
- [Sevc94] K.C. Sevcik, "Application Scheduling and Processor Allocation in Multiprogrammed Parallel Systems", Performance Evaluation, 1994.
- [Silb92] A. Silberschatz, J. Peterson, P. Galvin, Operating System Concepts, 3rd edition, Addison-Wesley, 1992.
- [Ste95] R. Steinmetz, "Analyzing the Multimedia Operating System", IEEE Multimedia, Spring 1995, pp. 68-84.

Appendix A Running a Simulation

All experimental results were generated using a computer program which simulates a multiprogrammed parallel computing environment with I/O devices. Inputs to the program consist of system, job model and simulation parameters while outputs include mean job response time, its confidence interval.

A structural and data flow diagram of a batch simulation run is illustrated in Figure A-1. The simulation program input parameters (physical system, job model and simulation

FIGURE A-1. Flow diagram of a batch simulation run

parameters) are entered by the user into a data file called 'test.dat' (see Figure A-2). The user then edits a batch variables file called 'bat.vars' to specify the length of the simulation, the length of the initialization period, what the X-axis and Y-axis represent and what are their values. The capability to specify a third variable parameter to represent the Z-axis and its values allows a user to run a simulation for different data replication policies or scheduling policies, for example, all in one batch run (see Figure A-3). To permit run-time

substitution of values for the X-axis and Z-axis, names are given to these axis using the variable `'_parmname'`. Note that the `'_parmname'` values are exactly the same values as in `'test.dat'`. The user then invokes the batch script file called `'bat'` which reads `'test.dat'`, converts the data into a form usable by the simulation program, and then runs the simulation program for each point on the X-axis and Z-axis. When the simulation completes, this batch script file takes the output files (`'test.out_'`) generated by the simulation program, extracts the desired simulation results using an AWK script and puts them into data files (`'test.plot_'`) in a format compatible with the GNUPLOT graphing program. The simulation results may be viewed graphically by loading the `'test.gnu'` file from within GNUPLOT.

Before starting a batch run, the following files (or links to these files) must be in the current working directory:

- `bat` - this is the name of the main script file. It is pretty generic and is used for all simulation runs. It reads `'bat.vars'` to configure itself for a particular run.
- `bat.vars` - this is the batch variables files which specifies what the batch run is trying to calculate. The X-axis, Z-axis values are given here, plus the appropriate names corresponding to what the X-, Z-axis represents. The simulation length and simulation initialization period is specified here. This file is read by `'bat'`.
- `genesis` - this is the PARASOL based simulation program which does the work.
- `test.dat` - the parameters for the system that is to be simulated. The X-, Z-axis values in `bat.vars` replace the corresponding values in this file.
- `awk` - this program found on most UNIX systems extracts the desired data from the simulation output files.

To start a batch, type:

```
>bat .
```

FIGURE A-2. Sample input parameter data file 'test.dat'

```

20 # Total CPUs
4 # Total IO Devs
1.0 # CPU Speed
1.0 # IODev Speed Ratio
2 # IODev MST Var
0.01 # IODev MST
1.0 # IODev MST CV
.95 # IODev MST Smaller
1 # IODev MTT Var
1.0 # IODev MTT CV
0.95 # IODev MTT Smaller
0 # Phys Spare1
0 # Phys Spare2
0 # Phys Spare3
0 # Phys Spare4
0 # Phys Spare5
0.001 # Job Arrival Rate
0.5 # IO Factor
5 # CPU Per Job
4 # IO Model
0 # IO Sched Policy
0 # IO Sched Subpolicy
2 # Data Rep Policy
10 # Num Blocks
900000. # Init Period
2 # Dev Alloc Policy
0 # Sim Spare1
0 # Sim Spare2
0 # Sim Spare3
1 # Num Child Var
10 # Mean Num Child
0.0 # Num Child CV
0.0 # Num Child Smaller
2 # Mean CPUB Var
1.0 # Mean CPU Burst
1.0 # Mean CPUB CV
0.95 # Mean CPUB Smaller
1 # Mean Total IOD Var
1.0 # Mean Total IOD CV
0.95 # Mean Total IOD Smaller
2 # Mean IOB Var
1.0 # Mean IOB CV
0.95 # Mean IOB Smaller
5 # O2 IO Burst
0.5 # O2 Read Prob
20 # NOP IO Threads
0 # IOM Spare1
0 # IOM Spare2
0 # IOM Spare3

```

```

0 # IOM Spare4
0 # IOM Spare5
0.00 # debug dump start time
977.0 # debug dump end time

```

FIGURE A-3. Sample batch variables file 'bat.vars'

```

# These are the things which need to be modified on a r r run basis.
# Note: simlen[cnt]: good initial value is 2,000,000
# initlen[cnt]: good initial value is 200,000
# - should normally be set to 1 batch length.
#
set simlen=(2000000. 2000000. 2000000. 2000000. 2000000. 9000000.
9000000. 9000000.)
set initlen=(200000. 200000. 200000. 200000. 200000. 900000. 900000.
900000.)
set numbatches=10
set xval = (0.001 0.1 0.25 0.35 0.40 0.45 0.50 0.55)
set zval = (0 1 2 3 4)
set xparname="Job Arrival Rate"
set yparname="Mean Job Completion Time"
set zparname="IO Sched Policy"
set pkeynames=(JBNOT JBOJF JBLCIO JBSCIO JBRR)
set headings='# Lambda JCT JCT-95%CI %ERR JQT JRT IOW SJS'

```

Appendix B Bi-phase Hyperexponential Distribution

The coefficient of variation of a random variable, X , is defined by the following equation where $V(X)$ is the variance of X and $E(X)$ is the expected value of X :

$$CV(X)^2 = V(X)/[E(X)]^2$$

Where service times of a facility has a standard deviation greater than the mean and therefore $CV > 1$, the hyperexponential distribution is a good model to represent the facility because this distribution can achieve any value of CV greater than unity. A hyperexponential distribution can be modeled in terms of a set of parallel exponentially distributed service facilities where only one facility can be active at any time. This representation based on exponentially distributed facilities makes the hyperexponential distribution very convenient to use. A bi-phase hyperexponential distribution refers to the situation where the set has two service facilities such as that shown below:

For the two stage model above, the mean of the exponentially distributed service time of facility 'i' is represented by S_i and the service time random variable is denoted X_i . The expected value of X_i is $E(X_i) = S_i$ and the variance is $V(X_i) = S_i^2$. The random variable X for

a service time is a random variable with hyperexponential distribution with parameters (S_1, S_2, p) where p is the probability of selecting server number 1, the server with the smaller mean ($S_1 < S_2$). The expected value and the variance of X can be shown to be equal to the following [Bank84]:

$$E(X) = pS_1 + (1-p)S_2$$

$$V(X) = 2pS_1^2 + 2(1-p)S_2^2 - [pS_1 + (1-p)S_2]^2$$

The overall expected value is $E(X)$ and the coefficient of variation is $CV(X)$. Many values of S_1 , S_2 and p can give the same values of $E(X)$ and $CV(X)$. For any given values of $E(X)$ and $CV(X)$, pick a value of p and then the means of the two distributions, S_1 and S_2 , can be calculated as follows [Koba81]:

$$S_1 = \text{SQRT}[(CV(X)^2 - 1) / (2p(1-p))] * E(X) * (p-1) + E(X)$$

$$S_2 = \text{SQRT}[(CV(X)^2 - 1) / (2p(1-p))] * E(X) * p + E(X)$$

In the simulation program, the $E(X)$ and the $CV(X)$ are the given quantities. The probability of selecting the equation with the smaller mean (ie. facility 1) is fixed at $p=0.95$. A random number is drawn to determine if facility 1 or 2 is selected. Then the corresponding mean for that facility is calculated and the mean is used to generate the exponentially distributed random variable for the service time of that facility, X_i .

Appendix C Supplemental Experimental Results

This appendix contains additional plots of experimental data to reinforce the findings presented in the main text. They are omitted from the main text as their presence is felt to clutter the main body of text. Instead, references to these graphs are made in the appropriate sections. The information provided by these graphs are supplemental in nature, and are not intended to present ideas not covered already in the main body of the thesis.

FIGURE C-2. R vs Number of NOP parallel I/O operations at low F_i . FIGURE C-3. Performance of job models at higher F_i .

FIGURE C-4. Device allocation policy performance for O1 workloads

$M=4, P=20, d_s=0.01, CV_s=1.0, n_d=10, d=1.0, CV_d=1.0, F_i=0.5, CV_i=1.0, \lambda=0.4.$

FIGURE C-5. R vs λ for NOP workloads, Random allocation, large $n_i=20$

$M=4, P=20, I=4, d_s=0.01, CV_s=1.0, n_d=10, d=1.0, CV_d=1.0, CV_i=1.0, F_i=0.5, n_i=20.$

FIGURE C-6. R vs CV_1 for NOP workloads and Random device allocation

$M=4, P=20, l=4, d_s=0.01, CV_s=1.0, n_d=10, d=1.0, CV_d=1.0, F_1=0.5, \lambda=0.4.$

FIGURE C-7. R vs CV_1 for NOP workloads and Round Robin device allocation

$M=4, P=20, l=4, d_s=0.01, CV_s=1.0, n_d=10, d=1.0, CV_d=1.0, F_1=0.5, \lambda=0.4.$

FIGURE C-8. R vs F_i for NOP workloads, Random allocation, low CV_i .

$M=4, P=20, I=4, d_s=0.01, CV_s=1.0, n_d=10, d=1.0, CV_d=1.0, CV_i=1.0, \lambda=0.4.$

FIGURE C-9. R vs F_i for NOP workloads, Round Robin allocation, low CV_i .

$M=4, P=20, I=4, d_s=0.01, CV_s=1.0, n_d=10, d=1.0, CV_d=1.0, CV_i=1.0, \lambda=0.4.$

FIGURE C-12. R vs λ for NOP workloads, Replicated-C, 2 I/O devices FIGURE C-13. R vs CV_i for NOP workloads, Replicated-C, 2 I/O devices

FIGURE C-14. R vs λ for O1 workloads, FixedRW(RR), 4 I/O devices FIGURE C-15. R vs CV_i for O1 workloads, FixedRW(RR), 4 I/O devices

FIGURE C-16. R vs F_i for OI workloads, FixedRW(RR), 4 I/O devices FIGURE C-17. R vs λ for NOP workloads, FixedRW(RR), 4 I/O devices

END

1 8 0 6 9 7

FIN