High Performance Disk Array Architectures

YEUNG Kai-Hau, Alan

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN THE DIVISION OF INFORMATION ENGINEERING

GRADUATE SCHOOL

THE CHINESE UNIVERSITY OF HONG KONG

HONG KONG

1995

To

my mother C. M. Lai

and

my wife Wai-Ying

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to Prof. T. S. Yum. He was my final year project supervisor when I was in my undergraduate study. With his guidance I started to appreciate the joy of doing research. Several years later, I became his PhD student and have again the valuable opportunity to learn from him. I appreciate very much his guidance, enthusiasm, rigorous mind, wisdom, encouragement, and the countless hours of time given to me. He teaches me not only research skills, but most importantly the correct attitudes of doing research.

I would also like to deeply thank my eldest brother Prof. K. S. Yeung. He taught me electronics when I was in my primary study. Under his influence I have chosen electronics as the field of my life career since my teenage. I thank him for giving me many timely advice, especially in the early stage of my PhD study.

It is my pleasure to thank the Department of Electronic Engineering, City University of Hong Kong. I thank Prof. Y. W. Lam and Prof. P. S. Chung for granting me a long study leave in 1994. Special thanks are given to Prof. P. S. Chung and Dr. K. T. Ko for their support and encouragement throughout my study.

My family has contributed tremendously to my life during my years of study. My mother, Mrs. C. M. Lai, faithfully prays for my graduation. When my research was not going well, her unconditional love always gave me lots of self-confidence. I thank my sons, Ho Ching and Ho Yi, for giving me much joy and hope everytime when I returned home from work. My wife Wai Ying is the only person in my life whom I cannot express all my thanks and appreciation in words.

ABSTRACT

In this thesis, I/O system designs using high performance disk array architectures are discussed. The thesis consists of five main parts. Firstly, a novel technique called the **Selective Broadcast** technique is proposed for high speed data distributions. Selective Broadcast technique relies on disk arrays to provide the high data throughput required. The technique significantly reduces the response time of data retrievals when compared to the non-selective broadcast techniques. A complete analysis on the Selective Broadcast technique is given in this thesis. Secondly, a new disk array architecture called **Dynamic Multiple Parity (DMP) Disk Arrays** is proposed for serial transaction processing database systems. By adding extra redundancy, DMP Disk Arrays effectively reduce the blocking delays for "write" operations which is caused by busy disks. Analysis on DMP Disk Arrays using Markov model is performed. Thirdly, another novel disk array architecture called **Dynamic Parity Logging (DPL) Disk Arrays** is proposed for engineering database systems. Unlike normal disk array architectures, DPL Disk Arrays use dynamic parity sets to protect data from disk failures. We show through analysis and simulation that DPL Disk Arrays have better performance than conventional disk array architectures. DPL Disk Arrays also have the journalling capability which is desirable for engineering database systems. Fourthly, a performance analysis on mirrored disk array is presented. Previous analyses on mirrored disk arrays are mainly approximate analyses which ignore the fork/join synchronization of the mirrored disks. In this thesis, a mirrored disk array is modeled as a fork/join queueing system and an exact Markov Chain analysis on the system is performed to obtain the average I/O job delay. Lastly, a state reduction technique for the exact analysis on fork/join queues is proposed. The technique significantly reduces the complexity of such exact analysis by many orders of magnitude.

Chapter One

Introduction

1.1 The Information Age

Thirty years ago, computers were considered by most people to be mysterious machines. They were room-size mainframes operated by specially trained professionals. They were very expensive and used only by governments and very large corporations. In 1975, the first recognizable personal computer called the Altair 8800 kit appeared [1]. Costed at US$400, Altair 8800 kit was a do-it-yourself 8-bit system with 256 bytes of RAM. It was the first time in computer history that computers started to become "personal." Since then, we have seen computer power grows by many orders of magnitude [2]. Today, computers are simply another tool for doing work. A small personal computer has the processing power of a mainframe which was used thirty years ago. Techonological innovation is still driving the growth of computer power. We will see new CPU chips, new computer architectures and new software to appear from time to time.

Like the computer industry, the communication industry is growing and changing at a fantastic pace. The growth is mainly driven by the advances in optical technologies. The field of fiber optic transmission can be considered to begin at about 30 years ago. In 1966, Charles Kao realized that data can be transmitted over a narrow filament of glass fiber [3]. He did an extensive series of experiements to prove that this was so. Since then, we have seen communication speed to increase many times. Gigabit communication systems are commercially available today and we foresee terabit networks to appear not very long in the future.

The advances in computer and communication technologies is driving the world into an information age. With the development of high speed global data-exchange networks, also known as the Information Superhighway, users all over the world will be given an opportunity to access a vast wealth of information. There are many possible applications on information superhighway [4]. To name a few, these include videoconferencing, document sharing, multimedia E-mail, and video on demand (VOD). The information superhighway mainly uses fiber cables for communications. It provides the necessary bandwidth for all demanding applications such as those mentioned above. Users' workstations are connected to the information superhighway through many relatively slow speed networks. These slow speed networks mainly used copper wires for data communications. This is due to the vast investment on the local loops which is in use today, and on the coaxial cables which already wire up homes for cable TV. Conceptually dividing the information superhighway as one layer of networks and the slow speed networks as another layer gives us an architecture similar to that discussed in chapter 2 of this thesis. In the rest of this chapter, we discuss how technological advancements in computer and communications demand fast I/O system designs.

1.2 The Importance of Input/Output

Figure 1-1 shows a general model for conventional computers, or the Von Neumann-type computers. Programs and data are stored in the secondary storage and moved to the main memory via the controller before they are used by the processor. The conventional computers is known to be particularly suitable for numerical processing. This is because I/O operations are rare once the programs and data are loaded in the main memory. The I/O subsystem, which is usually much slower than the main memory, therefore does not slow down the whole system. However, the I/O subsystem becomes the performance bottleneck when I/O intensive applications such as database management systems (DBMS) are run. This calls for many researches on I/O architectures for DBMS [5]. With DBMS being implemented on high speed networks, the volume of user transactions will increase manyfold [6]. To eliminate the I/O bottleneck, some researchers even expected that a distributed database system on a high speed network has to be "memory-resident" [7].

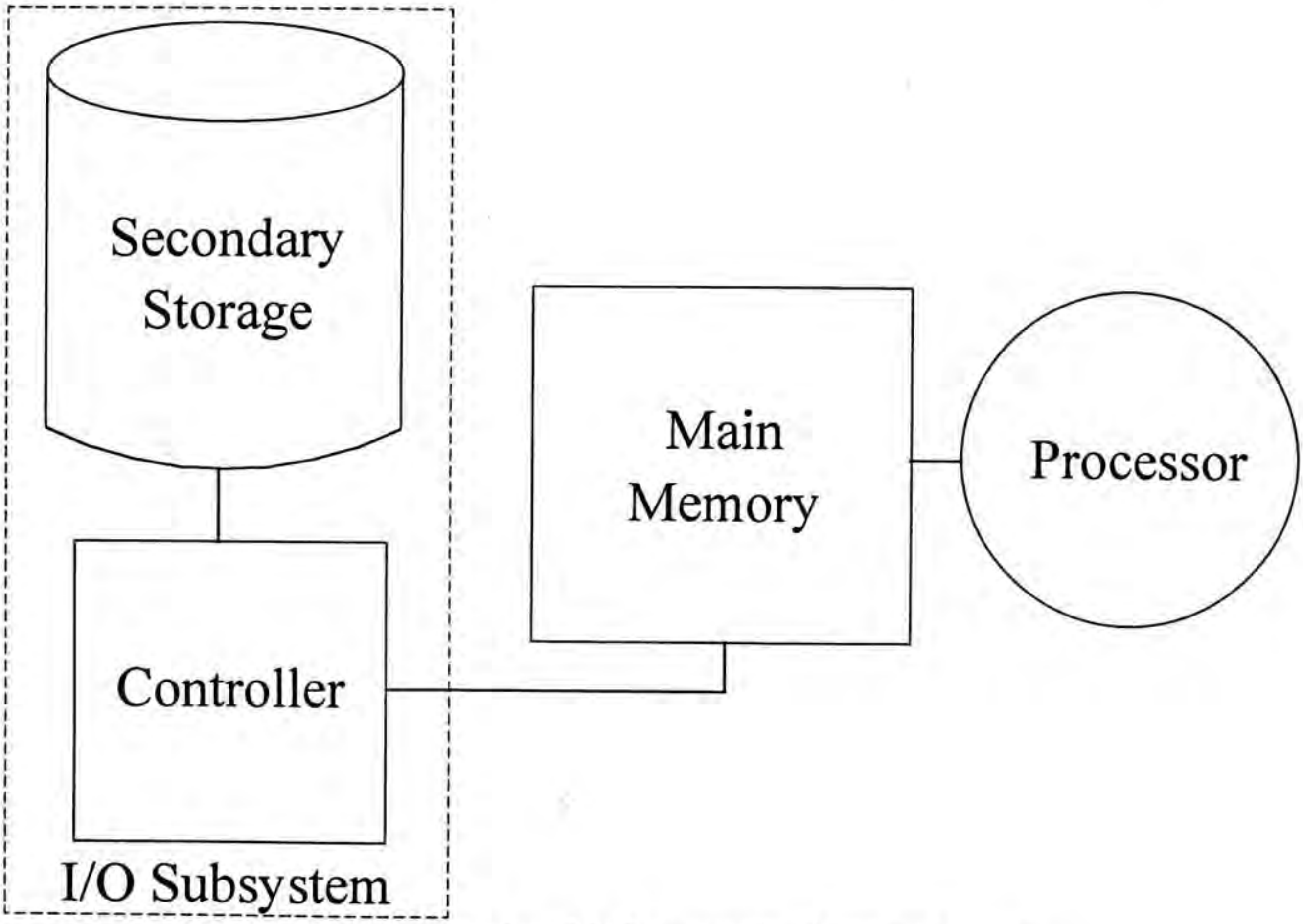


Figure 1-1 A general model of a conventional computer system.

Fast I/O systems are also demanded for systems having a huge information base. This includes all information systems with large amount of video or multimedia data. With the introduction of the information superhighway, remote access of large amount of data within a short time will become economically feasible. To provide enough data throughput for high speed data communications, information systems must have very fast I/O subsystems. Take interactive TV (ITV) as an example, it takes 95 TB to store the world's entire movie library in MPEG-2 format [pp.62, 8]. If this storage requirements of the ITV server seem daunting, the I/O is nightmarish. During peak hours in major cities, thousands of people may access a single ITV server requesting for videos. To satisfy the required data rate, the server should has an I/O throughput of several hundreds of megabytes per second. To further complicate the matters, the data rate of each user may change from time to time if VCR functions such as pause, rewind, fast-forward, slow motion, and frame advance are supported. For this reason Greg Hoberg, marketing manager of the video communications division at Hewlett-Packard, says concerning the ITV servers; "it's really an I/O machine" [pp.63, 8]. Because of this, researchers become aware of the fact that I/O performance becomes more or more crucial to the overall performance of a computer [9].

1.3 Redundant Arrays of Inexpensive Disks

In response to the increasing demands on fast I/O operations, researches on Redundant Array of Inexpensive Disks (RAID) were started by the end of 80's in the University of California at Berkeley [10,11]. The philosophy behind RAID is that instead of using a few expensive disks to achieve the performance and reliability required, many low cost disks working in parallel are used. With so many low cost disks, media availability becomes a serious problem. Parity encoding of data is used by RAID to provide high availability under disk failures. Six levels of RAID are defined [11,12] and they are briefly described below.

RAID Level 0: Disk array with no redundancy. Only data striping is supported in this level.

RAID Level 1: Mirrored disk array. Every data disk has a duplicate backup disk for reliability.

RAID Level 2 - Hamming coded disk array. A group of data disks has multiple parity disks and the number of parity disk is determined by Hamming code principles. Data is bit-interleaved across the data disks with additional parity bits stored in the parity disks.

RAID Level 3: Parity-protected disk array with bit-interleaved data. This level eliminates most of the overhead associated with error detection in RAID level 2. Level 2 uses parity checking to detect the erroneous disk and correct the wrong data. The error detection is not necessary because the disk controller is able to detect disk failures. Therefore, a single parity disk is enough for recovering the lost data in a failed disk. In level 3, data is also bit-interleaved across all data disks.

RAID Level 4: Parity-protected disk array with block-interleaved data and parity is stored in a dedicated parity disk. In levels 2 and 3, data is bit-interleaved across all data disks and therefore only one I/O operation can be performed at a time. In level 4, data is stored in a block-interleaved fashion. This enables multiple "read" operations to be performed simultaneously. A parity disk is used to store all the parity blocks. This parity disk becomes the performance bottleneck if "write" operations are frequent.

RAID Level 5: Parity-protected disk array with block-interleaved data and parity blocks are distributed across all disks. Since parity is distributed across all disks, parallel "write" operations can be performed.

Although commercial products on RAID are now available, new designs on RAID are still being investigated. Problems still exists in designing RAID for used in different applications. One major problem of RAID is that "write" operations are much slower than "read" operations when RAID is used in a DBMS. In solving this problem, we propose two novel disk array designs for two specific types of database systems. Another problem found in designing RAID systems is that it is very difficult to perform analysis on disk arrays [13]. For this reason, we discuss in chapters 5 and 6 the analysis on fork/join queues which is very useful for disk array performance study. Other issues found in RAID system designs include the design of RAID for continuous operations [14], the design of video systems using RAID [15], and many others. The discussions on these design issues, however, are beyond the scope of this thesis.

1.4 Outline of the Thesis

We have briefly discussed the developments of computer and communications technologies and their impact on the I/O system designs. We have also introduced RAID, an innovative ideas for speeding up I/O rates. In this thesis, research on I/O system designs is reported. There are five principle contributions reported in this thesis:

a) In solving the problem of efficiently distributing a huge amount of information to multiple users, a novel technique called **Selective Broadcast** technique for high speed data distribution is proposed [16-18]. Selective Broadcast systems rely on RAID to provide the necessary data throughput for data broadcasting. A complete analysis on the technique is performed and the technique is discussed in chapter 2 of this thesis.

b) In solving the problem of slow writing speed found in RAID, a novel architecture called **Dynamic Multiple Parity (DMP) Disk Arrays** is proposed for serial transaction processing systems [19]. Discussions on DMP Disk Arrays and the complete performance analysis on this architecture is given in chapter 3 of this thesis.

c) In solving the same problem stated in b), another new architecture called **Dynamic Parity Logging (DPL) Disk Array** is proposed for fast engineering datatabase systems [20]. Together with a complete performance analysis on the architecture, DPL Disk Array is discussed in chapter 4 of this thesis.

d) In chapter 5, performance analysis on mirrored disk array is discussed [21]. Average job delay for mirrored disk array is derived by modelling the system as an open fork/join queueing system.

e) In chapter 6 , we try to solve the problem of intractability found in the exact analysis on fork/join queues. A state reduction technique is proposed which significantly reduces the complexity of the analysis by many orders of magnitude [22]. The analysis on fork/join queues can be applied to the performance study of RAID systems.

The last chapter of this thesis discusses the possible extended research arising from the above studies.

References

[1] *Popular Electronics*, January, 1975.

[2] "20 years of Microcomputing," *PC Magazine*, December 20, 1994.

[3] C. K. Kao and G. A. Hockham, "Dielectric-fiber surface waveguides for optical frequencies," *Proc. IEE*, vol.113, no.7, pp.1151-1158, 1966.

[4] "Your Electronic Future," *Newsweek*, June 6, 1994.

[5] Stanley Su, *Database Computers*, McGraw-Hill, 1988.

[6] J. Gray, B. Good, D. Gawlick, P. Homan, and H. Sammer, "One Thousand Transactions Per Second," in *Proc. IEEE Spring Comput. Conf., COMPCON'85*, San Francisco, CA, February 1985, pp.96-101.

[7] S. Banerjee, Victor O. K. Li, and C. Wang, "Distributed Database Systems in High-Speed Wide-Area Networks," in *IEEE Journal on Selected Areas in Communications*, vol.11, no.4, pp.617-630, May 1993.

[8] Andy Reinhardt, "Building the Highway," *Byte*, pp.46-74, March, 1994.

[9] P. M. Chen and D. A. Patterson, "Storage Performance - Matrices and Benchmarks," *Proceedings of the IEEE*, Vol.81, No.8, pp.1151-1165, August 1993.

[10] G. A. Gibson, *Redundant Disk Arrays: Reliable, Parrallel Secondary Storage*, MIT Press, 1992.

[11] D. A. Patterson, G. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proc. of the ACM SIGMOD Conference on the Management of Data*, pp.109-116, 1988.

[12] P. M. Chen, G. A. Katz, and D. A. Patterson, "An Evaluation of Redundant Arrays of Disks Using an Amdahl 5890," in *Proc. SIGMETRICS*, pp.74-85, May 1990.

[13] E. K. Lee and R. H. Katz, "An Analytic Performance Model of Disk Arrays," *Proc. ACM SIGMETRICS*, pp.98-109, May 1993.

[14] M. Holland, and G. A. Gibson, "Parity Declustering for Continuous Operation in Redundant Disk Arrays," *ACM ASPLOS-V*, pp.23-35, 1992.

[15] F. A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAID^{TM} - A Disk Array Management System For Video Files," *ACM Multimedia 93*, pp.393-400,1993.

[16] T S Yum and K H Yeung, "The Confirm Before Delivery Technique for High Speed Data Distribution," *Proc. of IEEE GLOBECOM'93*, pp.1105-1109, November, 1993

[17] K H Yeung and T S Yum, "Selective Broadcast Data Distribution Systems," *Proc. of The 15th International Conference on Distributed Computing Systems*, pp.317-324, Vancouver, Canada, 1995.

[18] K H Yeung and T S Yum, "Selective Broadcast Data Distribution Systems," submitted to *IEEE Trans. on Computers* for possible publication.

[19] K H Yeung and T S Yum, "Dynamic Multiple Parity (DMP) Disk Array for Database Computers," submitted to *IEEE Trans. on Computers* for possible publication.

[20] K H Yeung and T S Yum, "Dynamic Parity Logging Disk Arrays for Engineering Database Systems," submitted to *IEEE Trans. on Computers* for possible publication.

[21] K H Yeung and T S Yum, "Performance Evaluation of Mirrored Disk Array," submitted to *IEEE Trans. on Computers* for possible publication.

[22] K H Yeung and T S Yum, "State Reduction in the Exact Analysis of Fork/Join Queueing Systems," submitted to *IEEE Trans. on Computers* for possible publication.

Chapter Two

Selective Broadcast Data Distribution Systems

This chapter describes a two tier architecture for high speed data distribution. The architecture consists of a database interface network which distributes information from a central database to a number of servers, and a user interface network which distributes information from the servers to the user terminals. The database interface network uses the **Selective Broadcast** technique to distribute data on a high speed channel. Data requested by users are filtered out by the servers and sent to the user terminals through the user interface network. The user interface network can be any conventional Local Area Network for connecting the servers and the user terminals. A very tight upper bound on the mean response time of the system for uniform request distribution is first derived. This is followed by an approximate analysis for general request distributions. Simulation results and design examples showed that Selective Broadcast technique can provide an order of magnitude smaller response time under normal traffic conditions when compared to the non-selective broadcast technique such as the Datacycle^{TM} system [11-12].

2.1 Introduction

Recent advances in computer and communication technologies have led to the development of information delivery systems that provide users with real time access to a broad spectrum of information. Examples of such information delivery systems are Videotex systems [1,2], multimedia information systems [3,4], digital news systems [5] and systems for medical imaging applications [6]. Conventional centralized information delivery systems are based on the central server model in which a central service computer replies to each user request in an individual response manner. The main drawback of this approach is the rapid increase of response time as the system load approaches the server's capacity. This situation can be improved with the introduction of the broadcast delivery and mixed delivery techniques [7-10]. In [10], it was shown that the response time using these techniques is significantly smaller than those based on the individual response model. However, the fact that a central server still remains in broadcast delivery models means that the limited power of the server is still the potential bottleneck of the overall information flow.

The basic configuration of such systems based on the central server model is shown in Figure 2-1. Information is organized into units called *pages*, and stored in a database. Users make requests and receive the requested pages through their terminals. The service computer retrieves the requested pages and transmits them to the user terminals via a communication network. Instead of considering database systems where there are many record updates, we consider only the read-oriented information delivery systems in this chapter.

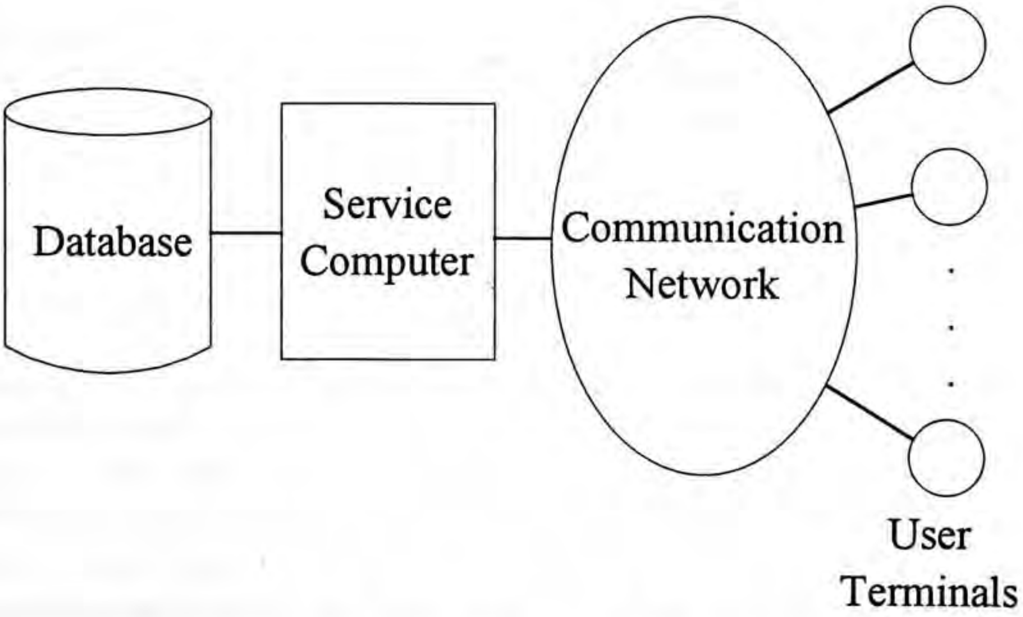


Figure 2-1 A typical information delivery system.

2.2 The Distributed Architecture

The system shown in Figure 2-1 has two potential bottlenecks: throughput of the communication network, and, I/O speed and processing power of the service computer. The relative significance of these two potential bottlenecks is application specific and depends on many factors. For future information systems with extremely large databases, the main bottlenecks will very likely be the service computer.

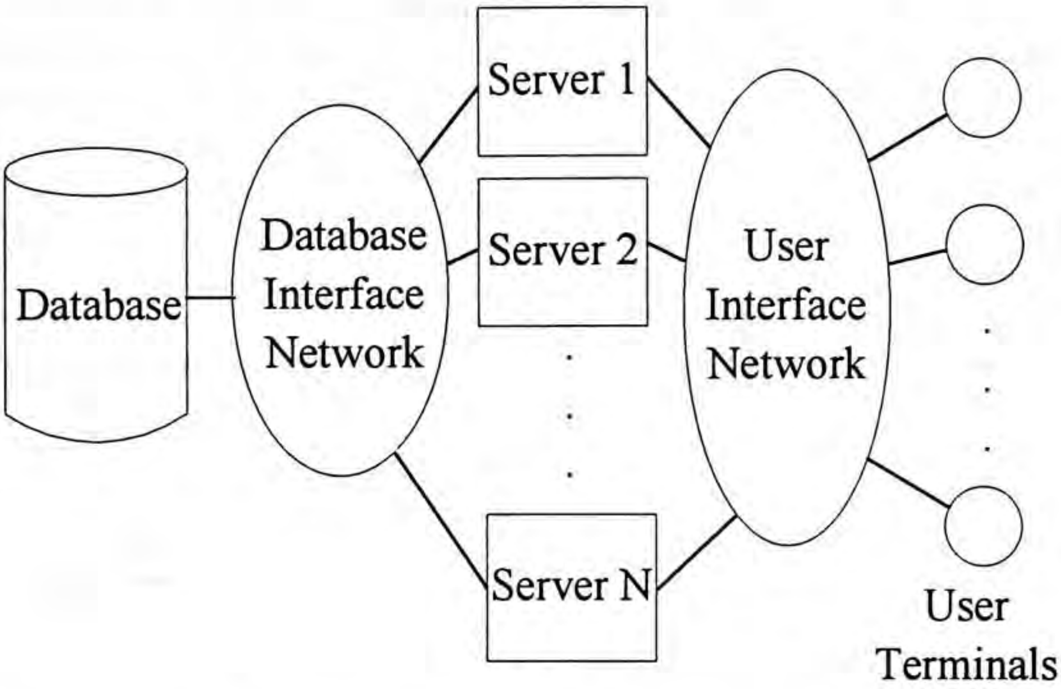


Figure 2-2 A modified configuration of information delivery system.

A multiple server architecture shown in Figure 2-2 is studied in this chapter. Here, the multiple servers working in parallel has the advantages of faster response due to distributed processing and modular growth in the number of servers. The database interface network and the user interface network parts are detailed as follows.

A) Database Interface Network

The function of the database interface network is to distribute information from the central database machine to the servers. One way to do this is to use the Datacycle^{TM} technique [11-12] whereby the entire database is pumped out from the database machine and distributed to the servers through a high speed link and the servers filter out the information required by the users. This technique has two advantages. First, it is relatively easy to optimize the I/O performance for sequential accesses. Second, the load on the database machine is independent of the volume of the traffic generated by the users. A performance analysis of Datacycle^{TM} and a new concurrency control scheme for such use is given in [13].

A new technique called the **Selective Broadcast** technique is proposed in this chapter for use in the database interface network to minimize the delay due to long cycle time of data. The data being broadcast is organized into units called *blocks*. Let there be a total of B blocks. The technique is based on the observation that in most cases only a small percentage of the data being pumped out is actually required; and so if a block is broadcast only when a confirmation for that block is received by the database machine, the data cycle time can be shortened to a small fraction of the original. The confirmation is done via the *Confirmation Ring* (Figure 2-3) which connects the database machine and all the servers. Periodically the database machine sends out a B bits frame to the ring. These B bits serves as a bit map of the B blocks of data. This frame circulates through all the N servers with a one bit delay on each. When the frame returns to the database machine, a new frame is sent for the next round of confirmation. A new frame generated by the database machine has a content of all zeros. If a server wants to confirm the i^{th} block (due to a request from a user it is serving, say), it simply write a "1" to the i^{th} bit of the frame. Otherwise the server simply passes the frame to the next server without modification. The content of the frame, therefore, reflects the specific blocks being confirmed.

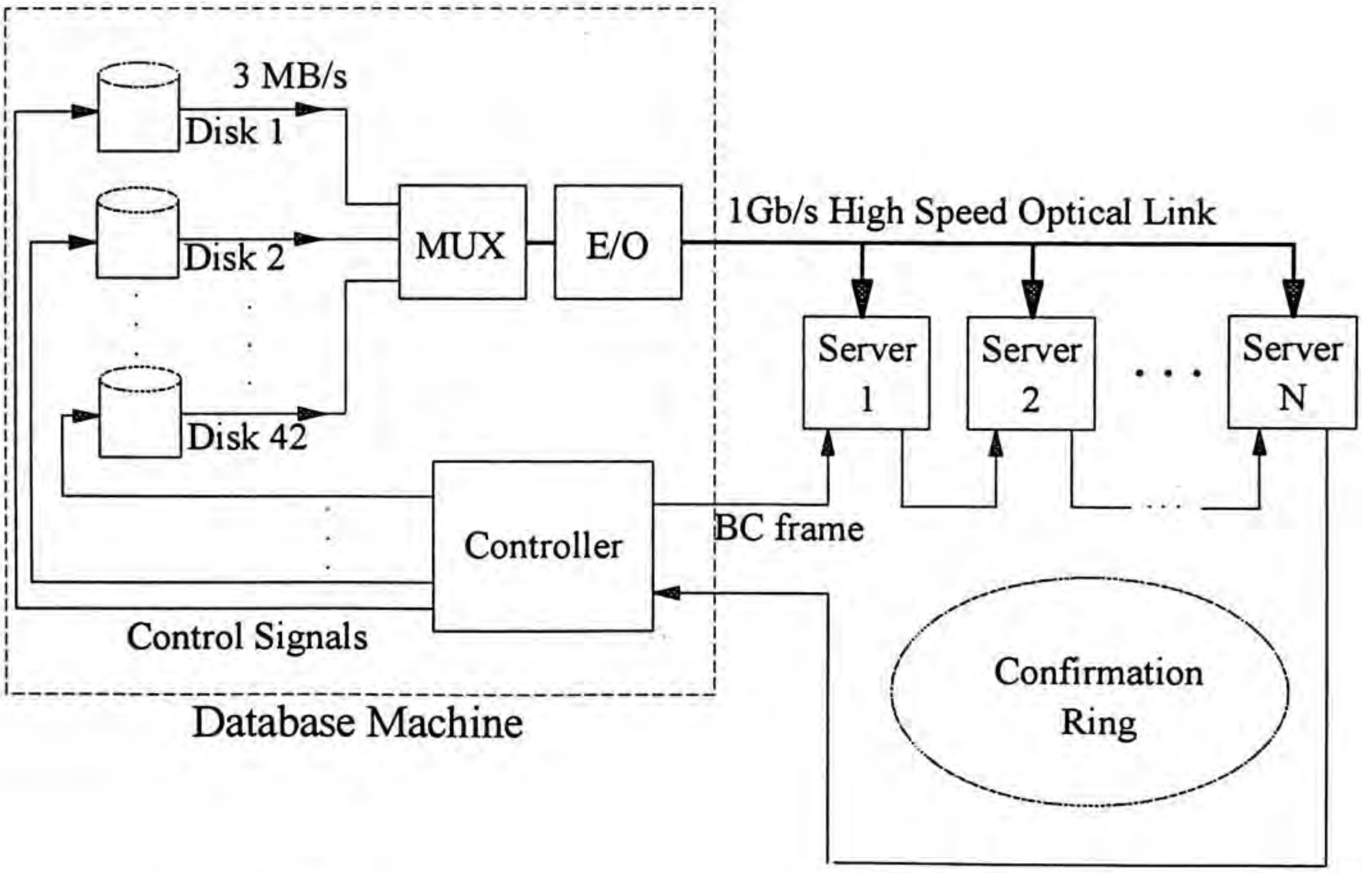


Figure 2-3 The database interface network and the central database machine.

At the database machine, the returned frame is copied into a B-bit *Block-confirm (BC) register*. The block in transmission is marked by a pointer on the BC register. Each time when the database machine is ready to send a block, it advances the pointer to the next non-zero bit and transmits the corresponding confirmed block. When the pointer reaches the end of the register, it cycles back to the beginning. Noting that the BC register contains the confirmation status of the entire database, so advance functions such as prefetching of data blocks from the database can also be incorporated. The analysis of such mechanisms, however, is beyond the scope of this chapter.

A large number of RAID architectures can be used to implement the database machine. A very simple example is shown in Figure 2-3 where 42 inexpensive disks of data rate 3 Mbyte/s are multiplexed onto a 1 Gb/s high speed optical link.

B) User Interface Network

The function of the user interface network is to exchange data between the servers and the user terminals. A large variety of LANs and MANs can serve this purpose. Discussion of their relative merit, however, is beyond the scope of this chapter.

C) Operation of the System

Figure 2-4 shows a block diagram of the distributed data distribution system. When a page request is issued by a user, it is sent to the request numbering box (RNB) through the request path. The RNB has the simple function of distributing the requests according to the processing rates of the servers. A server performs two types of operations as follows. At the user interface side, when a server receives a page request it uses its internal page directory to identify the particular block needed and makes a confirmation on that block. On the database interface side, a server filters out all its required blocks from the database interface network, extracts the requested pages from these blocks and sends them to the user terminals through the data path. The page directory is stored as a directory block in the database. When a server is powered up the directory block is first retrieved from the database and loaded into the server.

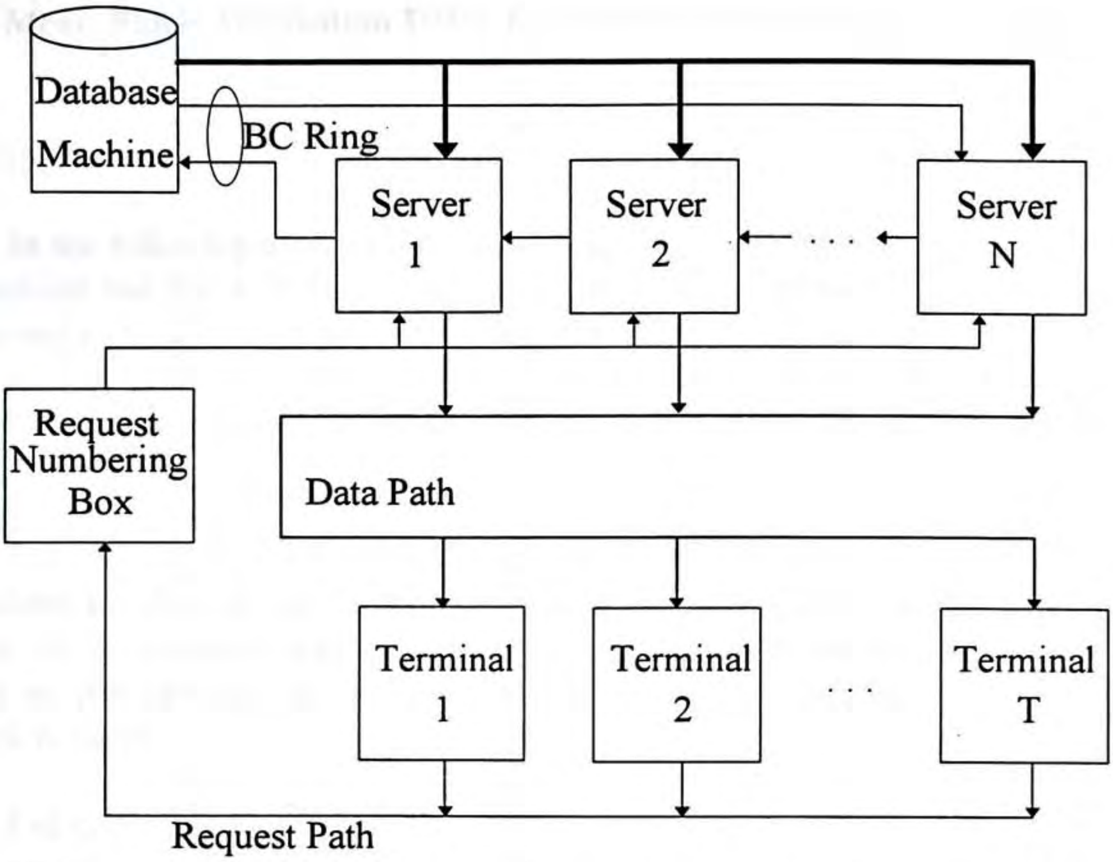


Figure 2-4 The distributed data distribution system.

2.3 Mean Block Acquisition Delay for Uniform Request Distribution[1]

A) Upper Bound Derivation

In the following analysis time is measured in *slots*, one slot being the time required to broadcast one *block* of data on the database interface network. The arrivals of requests is assumed to be a Poisson process of rate λ per slot. In this section we study the case where the request distribution is uniform, or the probability that a certain block is requested is identical for all blocks. The analysis for general request distributions will be given in the next section.

Let $\gamma(n)$ be the number of confirmed blocks waiting in the transmission queue at the database machine at slot n. $\gamma(n)$ therefore does not include the one in transmission. We shall, for convenience, call it the backlog size. Since the backlog size at slot n+1 depends only on $\gamma(n)$ and the number of requests in slot n, the evolution of $\gamma(n)$ is a discrete time Markov chain.

Let random variable A denote the number of requests per slot and random variable K denote the total number of blocks for which the A requests are located. As the requests are assumed to be randomly located in the database, the probability that a requests fall in k blocks is given in [15] as

$$P[K=k|A=a]=\mathrm{B}^{-a}\binom{\mathrm{B}}{k}\sum_{y=0}^{k}(-1)^{y}\binom{k}{y}(k-y)^{a} \tag{2-1}$$

Removing the conditioning on A, we obtain

$$P[K=k]=\sum_{a=0}^{\infty}P[K=k|A=a]\frac{\lambda^{a}e^{-\lambda}}{a!} \qquad k=0,1,2,\ldots,\mathrm{B} \tag{2-2}$$

Next, let random variable M_i be the number of arrivals whose requested blocks need to be confirmed when the backlog is i. We shall, for convenience, call these arrivals the *new* customers and those arrivals that do not generate confirmations the *subsequent* customers. If M_i=m, the backlog at the next slot will be i-1+m. Given that the backlog is i and k blocks are requested at the current slot, the probability that m out of these k blocks are to be confirmed is

[1] This section was reported in [14].

$P[M_i = m|K = k]$

$$= \frac{\begin{array}{c}(\text{number of ways to choose the } m \text{ request blocks from the } (\mathrm{B}\text{-}i) \text{ unconfirmed blocks})\\ \times(\text{number of ways to choose the remaining } k\text{-}m \text{ request blocks from the } i \text{ confirmed blocks})\end{array}}{(\text{number of ways to choose } k \text{ request blocks from B blocks})}$$

$$= \frac{\binom{\mathrm{B}-i}{m}\binom{i}{k-m}}{\binom{\mathrm{B}}{k}} \qquad m = 0,1,2,\dots,k$$

(2-3)

Removing the conditioning on *K*, we obtain the distribution of *new* customers in a slot as

$$P[M_i = m] = \sum_{k=0}^{\mathrm{B}} P[M_i = m|K = k]P[K = k] \tag{2-4}$$

At steady state, the transition probabilities are given by

$$h_{ij} \underline{\underline{\Delta}} P[\gamma(n+1) = j|\gamma(n) = i]$$
$$= \begin{cases} 0 & for\, j < i-1 \\ P[M_i = j-i+1] & for\, j \geq i-1 \end{cases} \tag{2-5}$$

Having obtained the transition probabilities, the equilibrium distribution of the backlog size, denoted as $\{\pi_0,\pi_1, \dots,\pi_B\}$ can be computed in the usual way. The average waiting time of the new customers, denoted by E[W_{new}], is given by the Little's formula as

$$E[W_{new}] = \frac{E[\gamma]}{E[M]} = \frac{\sum_{i=1}^{\mathrm{B}} i\pi_i}{\sum_{m=0}^{\mathrm{B}} m \sum_{i=1}^{\mathrm{B}} \pi_i P[M_i = m]} \tag{2-6}$$

Note that W_{new} is the waiting time experienced by the new customers. The subsequent customers will experience a smaller waiting time as the block request was already placed by the new customers. The expected waiting time of all customers E[W] can be computed as follow. Figure 2-5 shows the arrival of a new customer and its subsequent departure from queue after a waiting time W_{new} slots. During its stay in the queue, subsequent customers S_1, S_2, ..., S_j for the same block will arrive. The arrival rate is λ/B per slot. Let us condition on the event W_{new}=t. Since the arrival of the subsequent customers is a Poisson process, their arrival times are uniformly distributed in interval [0,t] and their average delay is just t/2. Let there be j such arrivals. Then the average waiting time of these j+1 customers is

$$E[W|j,t] = \frac{t + j\frac{t}{2}}{j+1} \tag{2-7}$$

Removing the conditioning on j, we have

$$E[W|t] = \sum_{j=0}^{\infty} \left(\frac{t + j\frac{t}{2}}{j+1} \right) \left(\frac{e^{-\lambda t/\mathrm{B}} (\lambda t / \mathrm{B})^{j}}{j!} \right)$$
$$= \frac{t}{2} + \frac{\mathrm{B}\left(1 - e^{-\lambda t/\mathrm{B}}\right)}{2\lambda} \tag{2-8}$$

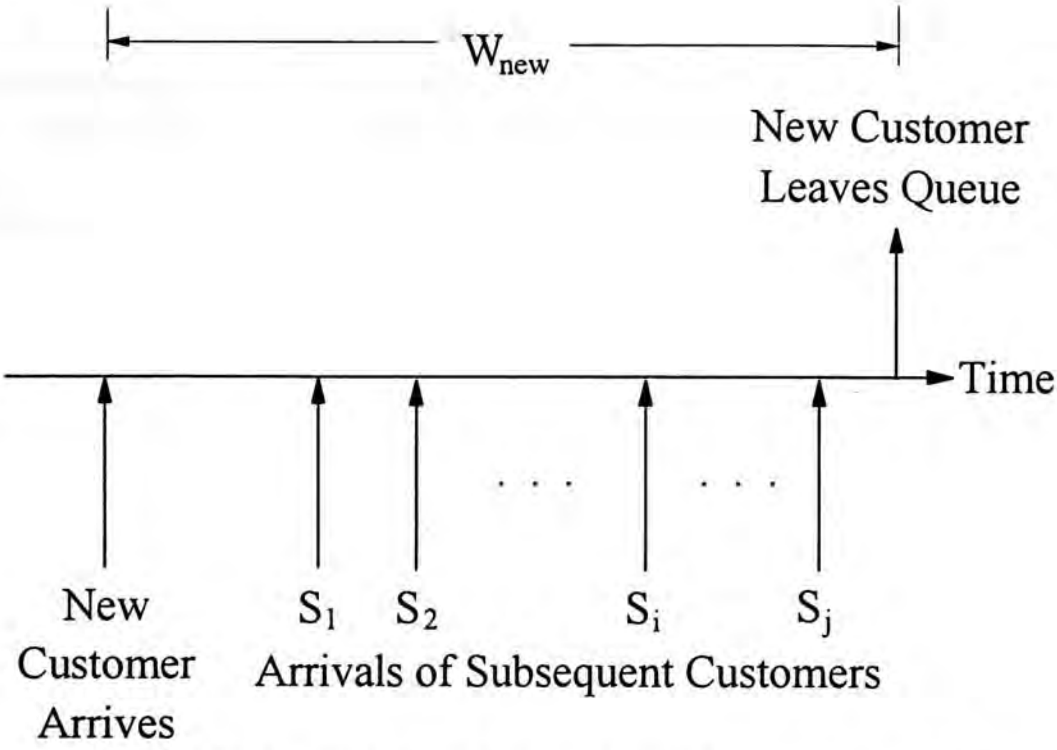


Figure 2-5 The arrivals of a new customer and the subsequent customers.

The evaluation of E[W] requires the distribution of W_{new} which is not available. But the use of Jensen's inequality [16] allows us to obtain an upper bound on E[W]. It is easy to show that (2-8) is a convex ∩ function of t and therefore the inequality gives

$$E[W] \le \frac{1}{2\lambda} \left\{ \lambda E[W_{new}] + \mathrm{B}\left(1 - e^{-\lambda E[W_{new}]/\mathrm{B}}\right) \right\} \tag{2-9}$$

A plot of (2-9) shows that the curve is fairly flat for typical values of B and λ. We would therefore expect the bound to be very tight. This is confirmed by the numerical results presented below. Finally the mean block acquisition delay E[T] is simply E[T]=E[W]+1.

After acquiring the blocks of data, the servers need to process them and deliver them to the users. The processing delay is usually a small fixed quantity independent of the system traffic. The delivery delay depends on the actual delivery network (usually a LAN) and its load. The mean response time for systems using Selective Broadcast technique is the sum of the three delays. The focus of the present study will only be on the mean block acquisition delay E[T].

B) Numerical Examples and Simulation Results

Numerical examples are given here to compare the average block acquisition delay for the Selective Broadcast technique and that for the Datacycle^{TM}. For Datacycle^{TM}, blocks are broadcast from the database machine sequentially with each block appearing exactly once in each cycle. The mean block acquisition delay for Datacycle^{TM} is simply (B/2)+1 slots and is independent of the request traffic.

Figure 2-6 shows the expected number of new arrivals per slot (i.e. E[M]) against the arrival rate for the five cases: B=40, 60, 80, 100 and 200. We observe that E[M] is practically independent of B. It grows linearly between $0 \le \lambda \le 1$ and saturates at 1 when $\lambda > 1$. This is expected because as shown in Figure 2-7 the mean backlog size is a very small fraction of B in the range $0 \le \lambda \le 1$ and so almost all arrivals are "new" customers.

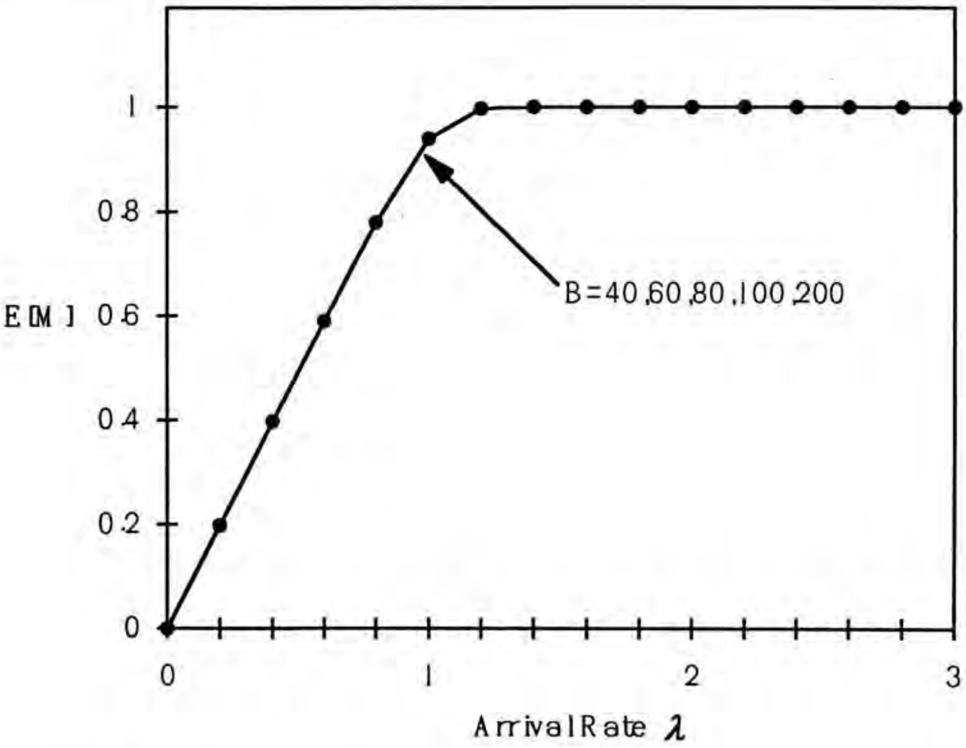


Figure 2-6 The expected number of new arrivals against arrival rate.

Figure 2-8 compares the upper bound of the mean block acquisition delay E[T] with the simulation results for Selective Broadcast technique when B=100. The 95% confidence intervals are all smaller than the size of the symbol "•" shown in the figure. The figure shows that the upper bound on E[T] is in fact very tight. Another observation is that the Selective Broadcast technique provides at least an order of magnitude smaller delay than that of the Datacycle^{TM} (which has a constant delay of 51) at the traffic level of $\lambda \le 0.8$. Selective Broadcast technique also provides uniformly lower delay than the Datacycle^{TM} under all traffic conditions. The results for B=200 is similar and therefore not shown.

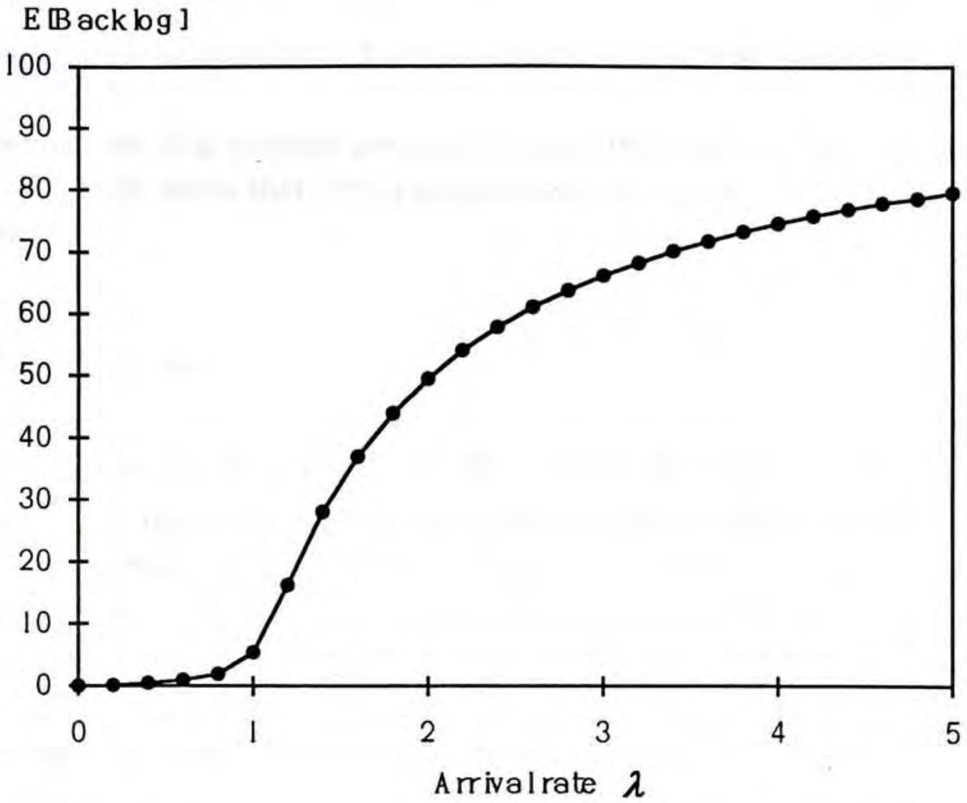


Figure 2-7 The mean backlog size against arrival rate for B = 100.

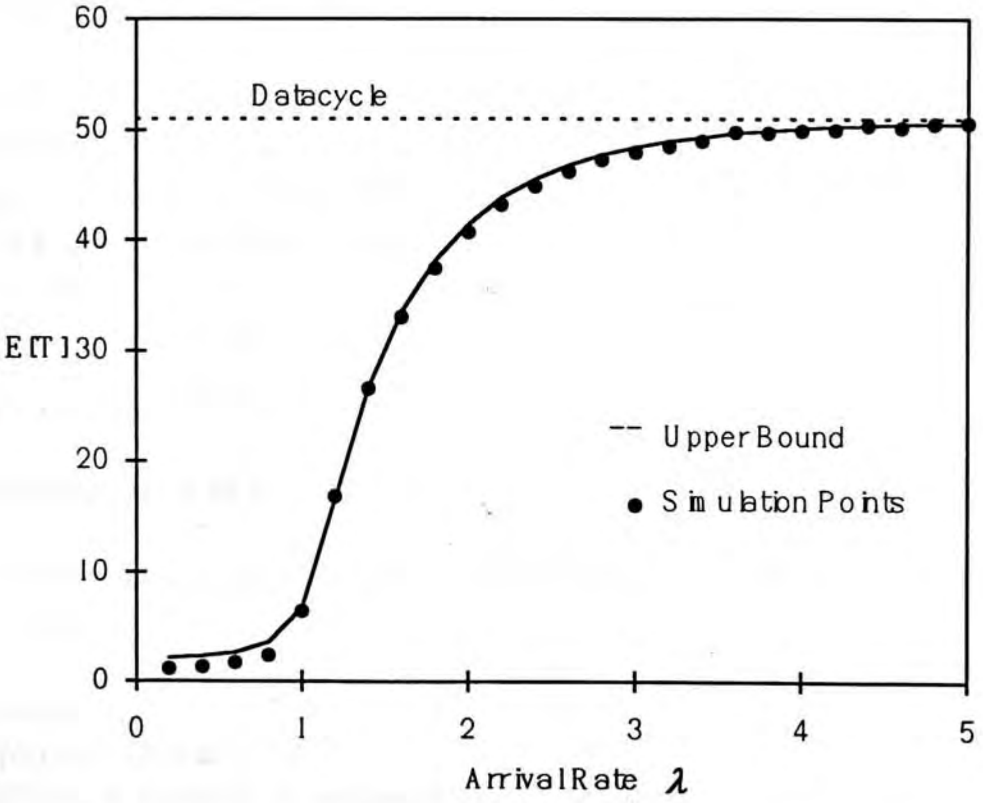


Figure 2-8 The upper bound on the mean block acquisition delay E[*T*] for Selective Broadcast technique when B = 100.

2.4 Mean Block Acquisition Delay for General Request Distributions

In this section we first present an approximate derivation of E[T] for general request distributions. We then show that the approximation is very accurate by comparing it with simulation results.

A) Approximate Analysis

Let λ_i be the Poisson arrival rate of the requests for block i, and $\sum_{i=1}^{B}\lambda_i = \lambda$. We define a *cycle* to be the period from the instance that a certain block has a chance to transmit until the instance it has the next chance to transmit. Obviously, blocks are not transmitted if they are not confirmed. Let random variable N be the cycle length in blocks and random variables n_i be the number of times block i appears in a cycle. Obviously, $N \geq 1$ for a cycle to exist and $n_i \in \{0,1\}$. We thus have $N=n_1+n_2+...+n_B$ under the condition that not all n_i's are zero. A bound on N without the attached condition is therefore

$$N \leq 1+n_1+n_2+...+n_B \qquad (2\text{-}10)$$

As the n_i's are all non-negative random variables, we can take expectation to obtain

$$E[N] \leq 1+\sum_{i=1}^{\mathrm{B}} P[n_i=1] \qquad (2\text{-}11)$$

where P[n_i=1] is the probability that block i is confirmed (i.e. will appear in a cycle) and is given by

$$P[n_i=1] = P[\text{at least one block } i \text{ arrival in a cycle}] = 1-e^{-\lambda_i N} \qquad (2\text{-}12)$$

Since the distribution of N is not available, the best we can do is to use Jensen's inequality [16] to obtain an upper bound on P[n_i=1]. It is easy to show that (2-12) is a convex $\cap$ function of N and therefore P[n_i=1] is bounded by

$$P[n_i=1] \leq 1-e^{-\lambda_i E[N]} \qquad (2\text{-}13)$$

Substitute into (2-11), we get

$$E[N] \leq (\mathrm{B}+1)-\sum_{i=1}^{\mathrm{B}} e^{-\lambda_i E[N]} \qquad (2\text{-}14)$$

Let g(E[N]) denote the R.H.S. of (2-14).

Lemma 1: *There is a unique solution denoted as x for E[N]=g(E[N]) in the interval (0,B+1).*

Proof: Figure 2-9 shows E[N] and g(E[N]), and we observe that

1. E[N]<g(E[N]) at E[N] =0.
2. g(E[N]) is a strictly increasing function of E[N].
3. E[N]>g(E[N]) at E[N]=B+1.

Therefore there exists one and only one solution in (0,B+1).

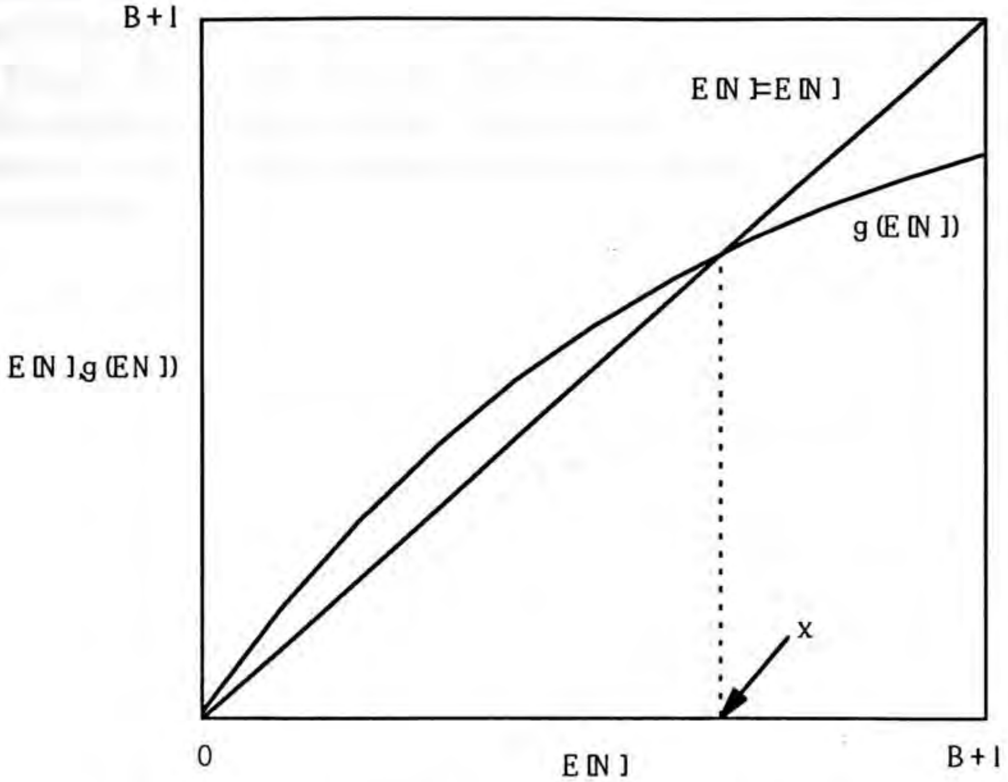


Figure 2-9 Relationship between E[N] and g(E[N]).

Theorem 1: *The expected number of confirmed blocks per cycle is bounded by x.*

Proof: Referring to Figure 2-9, we observe that E[N]≤g(E[N]) for E[N]≤x and E[N]>g(E[N]) for E[N]>x. Therefore, the inequality given by (2-14) is satisfied only when E[N]≤x. Substitute x into (2-14), we obtain

$$E[N] \leq (\mathrm{B}+1) - \sum_{i=1}^{\mathrm{B}} e^{-\lambda_i x} = x \tag{2-15}$$

Having obtained an upper bound on E[N], the mean block acquisition delay E[T] can be approximated by:

$$\begin{aligned} E[T] &= \text{average waiting time } + \text{ block transmission time} \\ &= \frac{x}{2} + 1 \end{aligned} \tag{2-16}$$

B) Numerical Examples and Simulation Results

To verify the analytical results obtained above, we perform simulations on three typical request distributions: uniform distribution, Zipf's distribution and geometrical distribution. The Zipf's distribution [17], stipulates that block i is requested with probability c/i where c is the normalization constant. For geometrical distribution, the request probability for block i is equal to $c\rho^i$, where c is the normalization constant and ρ is the skewing factor. We observe from the results shown in Figure 2-10 that for all three request distributions, the approximation on E[T] is very accurate compared to the simulation results. The results for the Zipf's and geometrical distributions show that Selective Broadcast technique performs better for more skewed distributions. This is expected because more requests are identifying on a smaller set of popular blocks with skewed distributions.

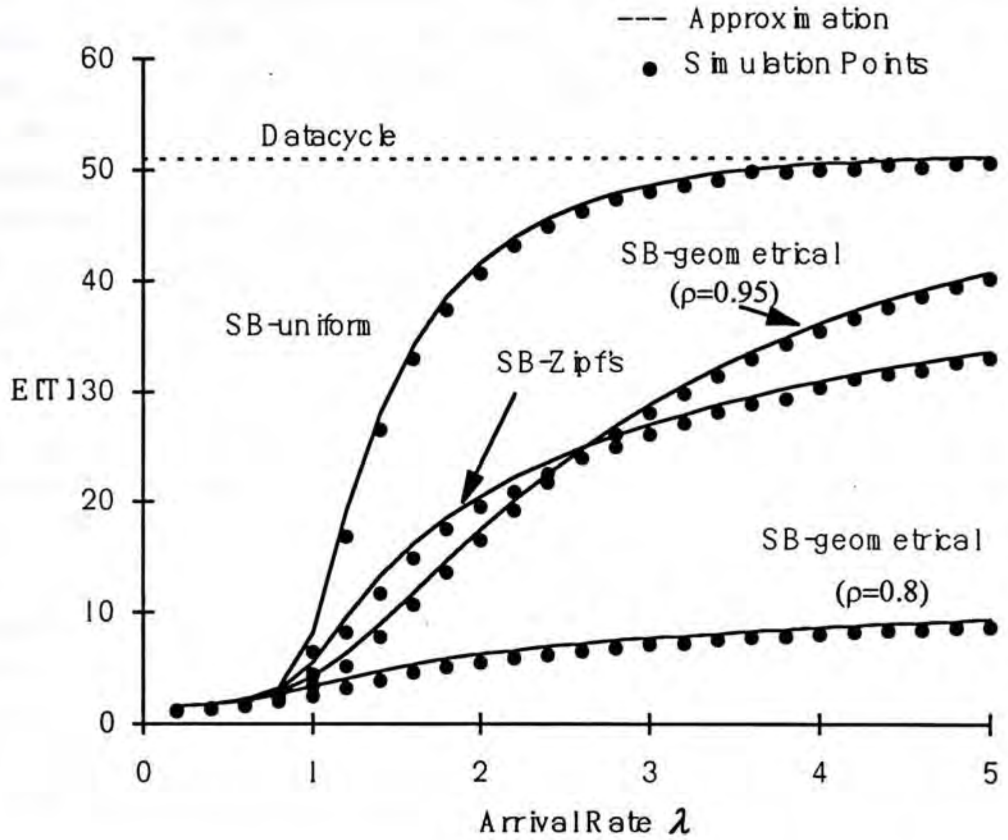
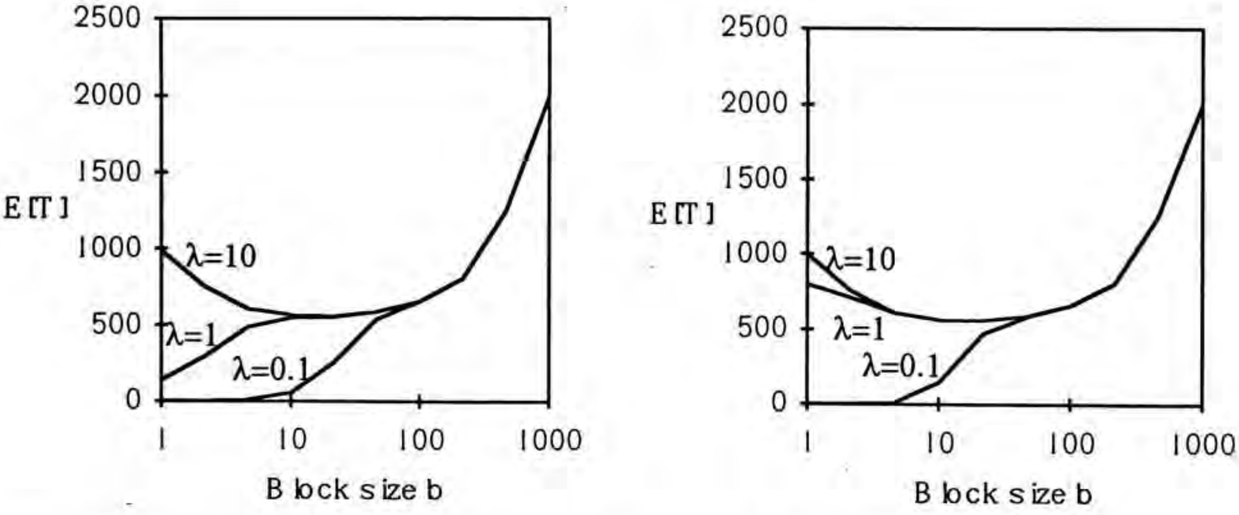


Figure 2-10 Approximation on the mean block acquisition delay E[*T*] for Selective Broadcast technique when B = 100.

2.5 Optimal Choice of Block Sizes

To determine the optimal block size, we redefine a slot to be the time required to broadcast one *page* (instead of one block) of data on the database interface network. Let b pages be grouped into a block and let the database has a total size of L pages. The number of blocks B is then equal to $\lceil \mathrm{L}/b \rceil$. The time for the database machine to locate a certain block on the disk is assumed to be a constant of d slots. With that the transmission time for a block is $b+d$ slots. One recent study on the HP-UX (Unix) computer systems shows that half of the I/O operations have nearly constant mean I/O time if cached disks are used [18]. Since the disk in a Selective Broadcast system operates sequentially with skips, I/O requests will therefore frequently hit the cache memory of the disk. The disk delay can thus be assumed to be closed to a constant for most I/O requests. With that, the approximate analysis given in the previous section applies directly by equating one "block" time unit to $b+d$ "page" time units.

Figure 2-11a shows how the block size b affects E[T] for uniform request distribution when $d=1$. At $\lambda=1$ and $\lambda=10$, b is optimal for a wide range between 5 and 100. When b is smaller than say 5, the disk delay will dominate the transmission overheads. On the other hand when b is larger than say 100, the time spent in broadcasting the non-requested pages becomes the major transmission overheads. At lower traffic level such as $\lambda=0.1$, the system always favors smaller b because the number of requested pages per confirmed block is closed to one. The time wasted in broadcasting other non-requested pages in a confirmed block will be minimized if a smaller b is chosen. In Figure 2-11b, similar conclusions can be drawn for the Zipf's distribution.



a) Uniform page request distribution.

b) Zipf's request distribution.

Figure 2-11 The effect on changing the block size b. The curves show the approximation on the mean block acquisition delay at different arrival rates.

2.6 Chapter Summary

In this chapter a new architecture for very high speed data distribution is proposed. The architecture consists of two separate networks: the database interface network and the user interface network. A new technique called the Selective Broadcast technique is used in the database interface network for high speed data distribution. An upper bound on the mean response time for uniform request distribution is derived and an approximate analysis for general request distributions is given. Simulation results show that the upper bound is very tight and the approximation is very good. Numerical examples show that the Selective Broadcast technique can give much smaller block acquisition delay than the Datacycle$^{\text{TM}}$ technique under non-overload conditions.

References

[1] J Gecsei, *The Architecture of Videotex System*, Englewood Cliffs, NJ: Prentice-Hall, 1983.

[2] M Sugimoto, M Taniguchi, S Yokoi, and H Hata, "Videotex: Advancing to Higher Bandwidth," *IEEE Communications Magazine*, pp. 22-30, February 1988.

[3] J H Irven, M E Nilson, T H Judd, J F Patterson, and Y Shibata, "Multimedia Information Services: A Laboratory Study," *IEEE Communications Magazine*, pp. 27-44, June 1988.

[4] J Rosenberg, R E Kraut, L Gomez, and C A Buzzard, "Multimedia Communications for Users," *IEEE Communications Magazine*, pp. 20-36, May 1992.

[5] E M Hoffert and G Gretsch, "The Digital News System at EDUCOM: A Convergence of Interactive Computing, Newspapers, Television and High-Speed Networks," *Communications of the ACM*, pp. 113-116, April 1991.

[6] J Kohli, "Medical Imaging Applications of Emerging Broadband Networks," *IEEE Communications Magazine*, pp. 8-16, December 1989.

[7] J W Wong & Mostafa H. Ammar, "Response Time Performance of Videotex Systems," *IEEE Journal on Selected Areas in Communications*, Vol. SAC-4, No. 7, October 1986.

[8] Mostafa H. Ammar, "Response Time in a Teletext System: An Individual User's Perspective," *IEEE Transactions on Communications*, Vol. COM-35, No. 11, November 1987.

[9] D K Gifford, J M Lucassen & S T Berlin, "The Application of Digital Broadcast Communication to Large Scale Information Systems," *IEEE Journal on Selected Areas in Communications*, Vol. SAC-3, No. 3, May 1985.

[10] John W Wong, "Broadcast Delivery," *Proceedings of the IEEE*, Vol. 76, No. 12, December 1988.

[11] Gary Herman, Gita Gopal, K C Lee & Abel Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems," *Proc. ACM SIGMOD*, May 1987.

[12] Tom Bowen, Gita Gopal, Gary Herman and William Mansfield, "A Scale Database Architecture for Network Services," *IEEE Communications Magazine*, Vol.29,

No.1, January 1991.

[13] S Banerjee, V O K Li and C Wang, "Distributed Database Systems in High-speed Wide-Area Networks," *IEEE Journal on Selected Areas in Communications*, vol. 11, pp.617-630, May 1993.

[14] T S Yum and K H Yeung, "The Confirm Before Delivery Technique for High Speed Data Distribution," *Proc. of IEEE GLOBECOM'93*, pp.1105-1109, November 1993.

[15] William Feller, *An Introduction to Probability Theory and Its Applications, 3/Ed, Vol I*, pp.60, John Wiley and Sons, 1968.

[16] R J Mc Eliece, *The Theory of Information and Coding*, Addison-Wesley, 1977.

[17] G K Zipf, *Human Behaviour and the Principle of Least Effort*, Addison-Wesley, 1949.

[18] C Ruemmler and J Wilkes, "An Introduction to Disk Drive Modeling," *IEEE Computer*, Vol.27, No.3, pp.17-28, March 1994.

Chapter Three

Dynamic Multiple Parity (DMP) Disk Arrays

The performance of today's database systems is usually limited by the speed of their I/O devices. Fast I/O systems can be built from an array of low cost disks working in parallel. This kind of disk architectures is called RAID (Redundant Arrays of Inexpensive Disks). RAID promises improvement over SLED (Single Large Expensive Disks) in performance, reliability, power consumption, and scalability. However, a general fact about RAID is that the "write" operation is difficult to speedup. In this chapter, we propose a new RAID architecture called **Dynamic Multiple Parity (DMP) Disk Array** for serial transaction processing database systems. Serial transaction processing database systems include engineering database systems, fully-replicated database systems using a completely centralized algorithm, and distributed systems using the conservative timestamp ordering algorithm. DMP Disk Array can significantly increase the I/O throughput by incorporating multiple parity disks. Due to the inherent distributed sparing property, DMP Disk Array can provide normal service to the users under single disk failure condition. Delay and maximum throughput analysis on DMP Disk Array is performed. Results show that DMP Disk Array can provide 30 to 40% improvement on "write" throughput over that of RAID level 5 when one extra parity disk is used.

3.1 Introduction

In the past decade, considerable attention has been drawn to the research and development of database computers. As Su [1] has stated in his book, there are three reasons why database computers are needed: 1) The need for efficient and effective data management; 2) The need for more powerful database management systems; and 3) High performance database computers become economically feasible due to the advancement in hardware technology and the reduction in hardware cost. The performance of a database system is usually limited by the speed of its storage devices. One good example is the Datacycle^{TM} project at BellCore starting from the late 80s [2,3]. BellCore proposed an innovative database architecture called the Datacycle^{TM}. In this architecture the entire database is periodically pumped out from the central database to a number of servers, in which the user required data are filtered out. Since the whole database is pumped out, Datacycle^{TM} has an unlimited throughput for read-only transactions. Datacycle^{TM} technique assumes that the database is "memory resident," i.e. the entire database has to reside in a very fast storage[1]. This assumption, however, limits its scope of applications.

Redundant Arrays of Inexpensive Disks (RAID) is an innovative concept in designing fast and reliable data storage systems. The philosophy behind RAID is that instead of using one single expensive disk to achieve the performance and reliability required, an array of low cost disks working in parallel are used. Five levels of RAID have been defined when RAID was first introduced[2] and the RAID level 5 was found to be one of the best [4]. For all levels of RAID the "write" operations are much slower than the "read" ones. This limitation is particularly severe for applications with frequent data updates.

There are two reasons why a "write" operation takes more time for RAID. Firstly, a "write" operation involves the additional step of reading back the old data from one disk and parity from another disk. Secondly, a "write" operation involves the waiting time for two specific disks to be free simultaneously before actual writing. This waiting time can be reduced by using the technique presented in this chapter.

In this chapter we propose a new RAID architecture called Dynamic Multiple Parity (DMP) Disk Array for serial transaction processing database systems. Many database systems process transactions in this way. Examples are engineering database systems [5-6], fully-replicated database systems using the completely centralized algorithm, and distributed systems using the conservative timestamp ordering algorithm [7-9]. When we discuss the operation of DMP Disk Arrays, we will see how this kind of database systems handle I/O requests in a way different from other database systems. We will also show that DMP Disk Array can significantly reduce the waiting time of disk operations, and provide a higher I/O throughput than the RAID level 5. In the next

[1] In the prototype built by BellCore, the whole database resides in RAM.

[2] Some disk manufactors introduce their own levels of RAIDs later.

section, we describe DMP Disk Array in detail. We then present average delay analysis in section 3.3 and maximum throughput analysis in section 3.4. In section 3.5, results on simulation with precise disk model is given. We then conclude the chapter in section 3.6.

3.2 DMP Disk Array

A. Sector Coordinate System

The sector coordinate system can be used to describe RAID operations, including that of DMP Disk Array. Consider a disk array system with *M* disks where each disk consists of *Z* sectors and each sector can store *K* bits of information (Figure 3-1). Let $S(i,j)=(b_1^{i,j},b_2^{i,j},b_3^{i,j},\ldots,b_K^{i,j})$ be the bit pattern of sector *j* in disk *i*. It can be data or parity. As an example, a RAID level 1 is described in this sector coordinate system as:

$$S(i,j) = S(i+1,j) \qquad j=1,2,3,\ldots,Z, \quad i=1,3,5,\ldots,M\text{-}1 \tag{3-1}$$

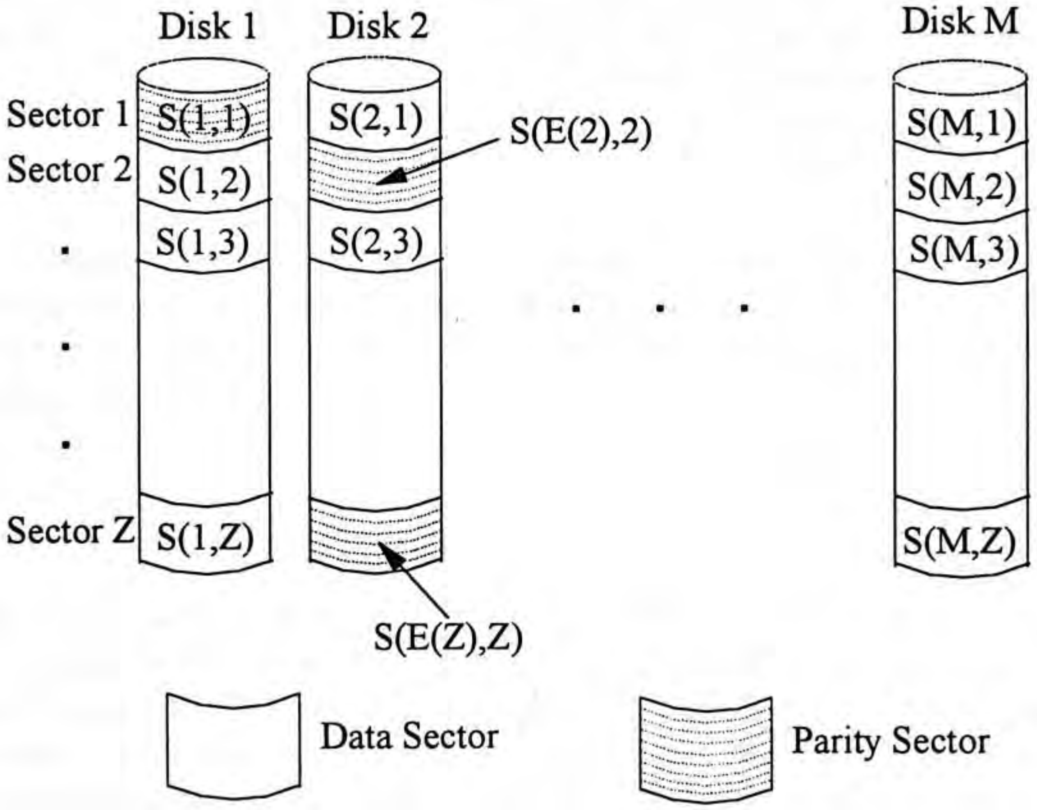


Figure 3-1 The sector coordinate system showing level 5 RAID.

Similarly, RAID level 4 and level 5 can be described as:

$$\sum_{i=1}^{M} S(i,j) = \begin{cases} (1,1,1,\ldots,1) & \text{for all } j \ \ \{\text{odd parity}\} \\ (0,0,0,\ldots,0) & \text{for all } j \ \ \{\text{even parity}\} \end{cases} \tag{3-2}$$

where Σ is defined here as the mod-2 sum of the sectors' contents:

$$\sum_{i=1}^{M} S(i,j) = S(1,j) \oplus S(2,j) \oplus \ldots \oplus S(M,j)$$

$$= (b_1^{1,j}, b_2^{1,j}, \ldots, b_K^{1,j}) \oplus (b_1^{2,j}, b_2^{2,j}, \ldots, b_K^{2,j}) \oplus \ldots \oplus (b_1^{M,j}, b_2^{M,j}, \ldots, b_K^{M,j})$$

$$= \left(b_1^{1,j} \oplus b_1^{2,j} \oplus \ldots \oplus b_1^{M,j}, b_2^{1,j} \oplus b_2^{2,j} \oplus \ldots \oplus b_2^{M,j}, \ldots, b_K^{1,j} \oplus b_K^{2,j} \oplus \ldots \oplus b_K^{M,j}\right) \tag{3-3}$$

Let $E(j)$ be the location of the parity sector at row j. For RAID level 4 $E(j)=M$ and for RAID level 5 $E(j)=(j \bmod M)$. The placement of these parity sectors are shown in Figure 3-1. A study on the various parity placement methods for RAID level 5 can be found in [10].

B. Sector Organization

The DMP Disk Array proposed in this chapter is a new RAID architecture for which the RAID level 5 is a special case. RAID level 5 has only one parity sector in each sector row and therefore it is not possible to simultaneously update two or more sectors on the same row. DMP Disk Array allows such updates by placing R parity sectors in each row. Their locations $E_1(j), E_2(j), \ldots, E_R(j)$ for row j are

$$E_r(j) = r + j \bmod M \quad r = 1,2,\ldots,R \tag{3-4}$$

Lee [10] showed that for relatively large request size of hundreds of kilobytes, the choice of parity placement can significantly affects the performance of disk arrays whereas for small request size, the choice of parity placement is insignificant to system performance. We therefore arbitrarily choose the parity locations as stated in (3-4). Although there are R parity sectors in each row, parity integrity described in (3-2) is always maintained for DMP Disk Array. (An example on the sector organization of DMP Disk Array for $R=2$ is shown in Figure 3-2.)

There are two advantages of placing R parity sectors in each row. First, for each data sector modification, we can choose any one of the R parity sectors in the same row for simultaneous parity modification. Hence blocking due to busy disks can be significantly reduced. Second, up to R data sectors on the same row can now be modified simultaneously. We now prove that DMP Disk Array has these useful properties.

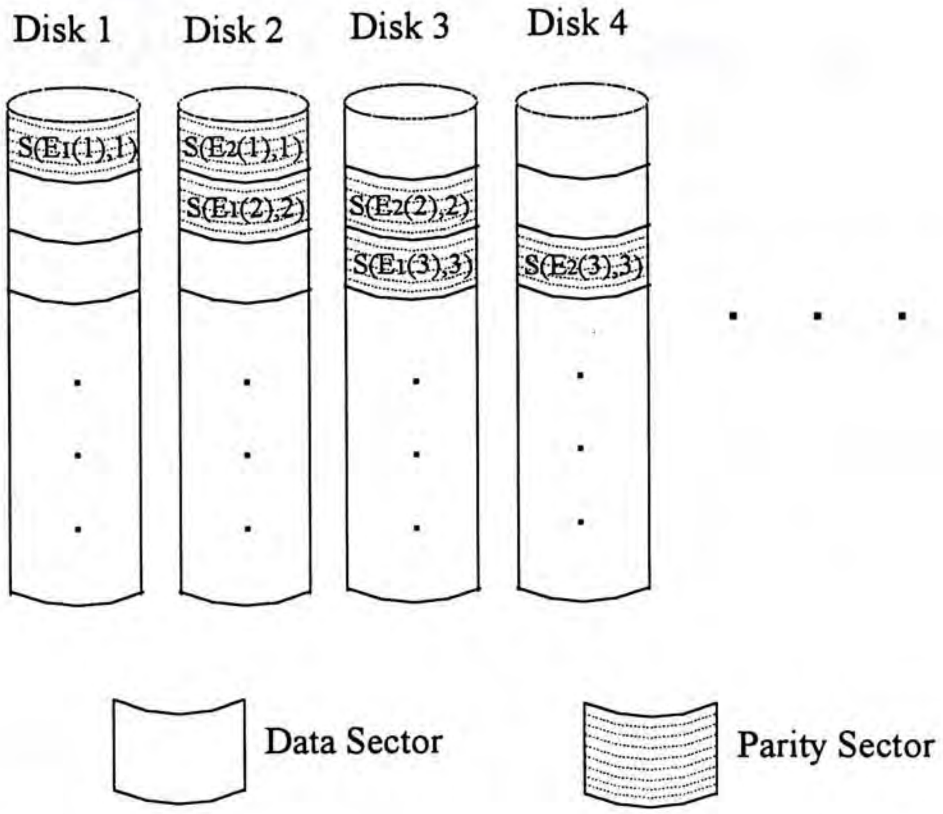


Figure 3-2 The placement of parity sectors for DMP Disk Array when $R=2$.

C. Properties of DMP Disk Array

Property 1 concerns about the simultaneous negation of two bits on the same row.

PROPERTY 1. *For any row j, simultaneous negation of any two bits in the same bit positions of two different sectors will not affect the parity sum of the row.*

PROOF. Consider the set of r^th^ bits of each sector in sector row j, i.e., b_r^{1j}, b_r^{2j}, ... and b_r^{Mj}. In order to maintain parity integrity, the sum of these bits should always be 1 for odd parity, or 0 for even parity. The parity sum after the negation of any two bits b_r^{aj} and b_r^{bj}, where $a \neq b$ is:

$$
\begin{aligned}
& b_r^{1,j}\oplus\ldots\oplus\overline{b_r^{a,j}}\oplus\ldots\oplus\overline{b_r^{b,j}}\oplus\ldots\oplus b_r^{M,j} && \\
&\quad =\overline{b_r^{a,j}}\oplus\overline{b_r^{b,j}}\oplus b_r^{1,j}\oplus\ldots\oplus b_r^{M,j} && \{\text{cummutative law}\} \\
&\quad =\left(\overline{b_r^{a,j}}\oplus\overline{b_r^{b,j}}\right)\oplus\left(b_r^{1,j}\oplus\ldots\oplus b_r^{M,j}\right) && \{\text{associative law}\} \\
&\quad =\left(\overline{\overline{b_r^{a,j}}}\,\overline{b_r^{b,j}}+\overline{b_r^{a,j}}\,\overline{\overline{b_r^{b,j}}}\right)\oplus\left(b_r^{1,j}\oplus\ldots\oplus b_r^{M,j}\right) && \{\text{definition of exclusive - or}\} \\
&\quad =\left(b_r^{a,j}\,\overline{b_r^{b,j}}+\overline{b_r^{a,j}}\,b_r^{b,j}\right)\oplus\left(b_r^{1,j}\oplus\ldots\oplus b_r^{M,j}\right) && \{\text{law of double negation}\} \\
&\quad =\left(b_r^{a,j}\oplus b_r^{b,j}\right)\oplus\left(b_r^{1,j}\oplus\ldots\oplus b_r^{M,j}\right) && \{\text{definition of exclusive - or}\} \\
&\quad =b_r^{a,j}\oplus b_r^{b,j}\oplus b_r^{1,j}\oplus\ldots\oplus b_r^{M,j} && \{\text{associative law}\} \\
&\quad =b_r^{1,j}\oplus\ldots\oplus b_r^{a,j}\oplus\ldots\oplus b_r^{b,j}\oplus\ldots\oplus b_r^{M,j} && \{\text{cummutative law}\}
\end{aligned}
\tag{3-5}
$$

and is identical to the parity sum before the simultaneous negation. We can extend the argument on the modification of two bits to that of the modification of two sectors on the same row. This is stated as Property 2.

PROPERTY 2 *Parity integrity of a row can be maintained by modifying any one of the parity sectors in the same row.*
Property 2 implies that there can be *R* different ways to update a data sector. Due to this flexibility, the probability that a "write" request is blocked due to the busy disk can be reduced.

PROPERTY 3 *Consider the simultaneous modification of data in sector j of disk a and parity in sector j of disk b. We denote the old data sector, the new data sector, the old parity sector and the new parity sector as $S(a,j)|_{old}$, $S(a,j)|_{new}$, $S(b,j)|_{old}$ and $S(b,j)|_{new}$ respectively. For maintaining data integrity, the new parity sector should be:*

$$S(b,j)|_{new} = S(a,j)|_{old} \oplus S(a,j)|_{new} \oplus S(b,j)|_{old} \tag{3-6}$$

PROOF In order to maintain parity integrity, the partial sum of the two sectors *S(a,j)* and *S(b,j)* must not be changed after sector modification, i.e.

$$S(a,j)|_{new} \oplus S(b,j)|_{new} = S(a,j)|_{old} \oplus S(b,j)|_{old} \tag{3-7}$$

Solving for $S(b,j)|_{new}$, (3-6) is obtained.

PROPERTY 4 *For DMP Disk Array with R parity sectors in each row, R data sectors locating at the same row can simultaneously be updated.*

PROOF Equation (3-7) shows that for any sector update on row *j*, the partial sum of the data sector and the parity sector will always be the same. Thus, the particular sector update will not affect the updating of the other sectors in row *j*. From property 3, we find that each sector update requires the old contents of two sectors only. Therefore, each update is actually carried out by two disks working in cooperation, and is independent of the operations of the rest of the disks. Therefore with R parity sectors, R simultaneous updates can be performed.

PROPERTY 5 *Consider a DMP Disk Array with M disks and R parity sectors in each row (R>1). When a disk fails, it is possible to reconfigure the remaining M-1 disks to a new array with R-1 parity sectors in each row without data loss.*

PROOF Let (p,p,...,p) be the parity sum of all the sectors in a row, say row j, and let $S(b_1,j),S(b_2,j),\ldots,S(b_R,j)$ be the parity sectors. Obviously, $b_1=E_1(j)$, $b_2=E_2(j)$, ..., $b_R=E_R(j)$. Suppose disk i fails. We consider two cases for the recovery of $S(i,j)$.

(i) If $S(i,j)$ is a data sector, it can be recovered from

$$S(i,j)=(p,p,\ldots,p)\oplus \sum_{n=1,n\neq i}^{M} S(n,j) \tag{3-8}$$

One of the parity sector, say $S(b_R,j)$, can be used to store the recovered data $S(i,j)$. To maintain parity integrity another parity sector, say $S(b_1,j)$, is modified as:

$$S(b_1,j)=S(b_1,j)\oplus S(b_R,j) \tag{3-9}$$

before storing the recovered data in disk b_R. The recovered DMP Disk Array has now R-1 parity sectors.

(ii) If $S(i,j)$ is a parity sector, then no data is lost. The parity integrity can be recovered by modifying another parity sector, say b, as follows:

$$S(b,j)=(p,p,\ldots,p)\oplus \sum_{\substack{n=1,n\neq i,\\ n\neq b}}^{M} S(n,j) \tag{3-10}$$

Property 5 tell us that DMP Disk Array has the distributed sparing property discussed in [11]. It is shown in [11] that distributed sparing is the best sparing technique for small disk arrays.

D. Principle of Operation

Figure 3-3 shows the organization of DMP Disk Array. Requests from host are directly sent to the disk controller for I/O operation. The disk controller consists of four parts: a FCFS queue, a local memory, a scheduling processor, and a DMA controller. The queue is used for storing I/O requests. Since we are considering systems which execute transactions in a strict order, a single global queue with FCFS service discipline is used. Note that this is different from other systems reported in the literature which use separate disk queues [12-15]. The data associated with each request (i.e. the new data for a sector) is stored in the local memory when the request is placed on the queue. The local memory is also used for buffering the data sent to/read from each of the disks with the help of the DMA controller. The DMA controller functions basically as a multiplexer/demultiplexer and handles simultaneous data transfers to various disks. The scheduling processor is responsible for distributing I/O requests to the disks, and performs all necessary processing. Specifically, its functions are outlined as follows:

1. It read a request from the FCFS queue when ready and determine whether the request is a "read" or a "write" type.

2. For a "read" request, the processor will
   i) check the status of the disk involved with this request;
   ii) instruct the disk involved to read the target sector;
   iii) load the sector to the local memory; and
   iv) signal the host for data ready.

3. For a "write" request, the processor will
   i) check the status of the disks and select at random the parity sector of a non-busy disk;
   ii) read the old data sector and the selected parity sector;
   iii) compute the new parity sector according to equation (3-7);
   iv) write the new data sector and the new parity sector to their corresponding disks;
   v) read back the parity and data sectors for verification; and
   vi) signal the host for "write" completion.

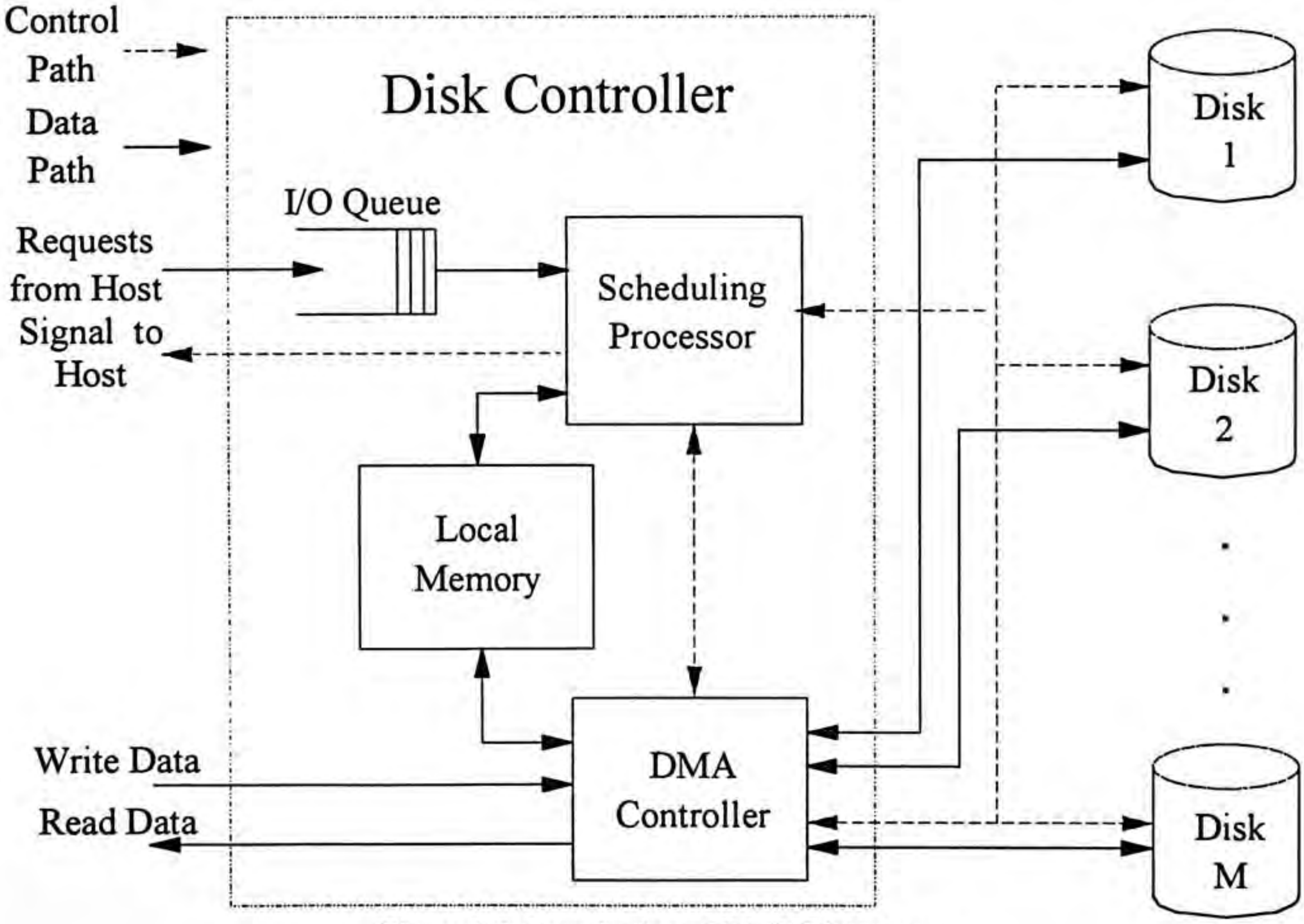


Figure 3-3 Organization of DMP Disk Array

3.3 Average Delay

A) Analysis

Figure 3-4 shows a queueing model for DMP Disk Array. Requests sent from host become *jobs* to be served in the servers. Job arrivals are assumed to be a Poisson process with rate λ. A job is of the "write" type with probability α and of the "read" type with the remaining probability. Jobs not yet served by the disk array are queued in a FCFS queue. We call the job which is at the top of the queue the *Head Of Line (HOL) job*. The probability that the HOL job needs to access a particular disk is assumed to be the same for all disks. This assumption is usually not true. But it can be made true by distributing the frequently accessed data uniformly across all the disks. For mathematical convenience we assume that the service time for a job is exponentially distributed with the service rates for a "write" job and a "read" job denoting as μ_w and μ_r respectively.

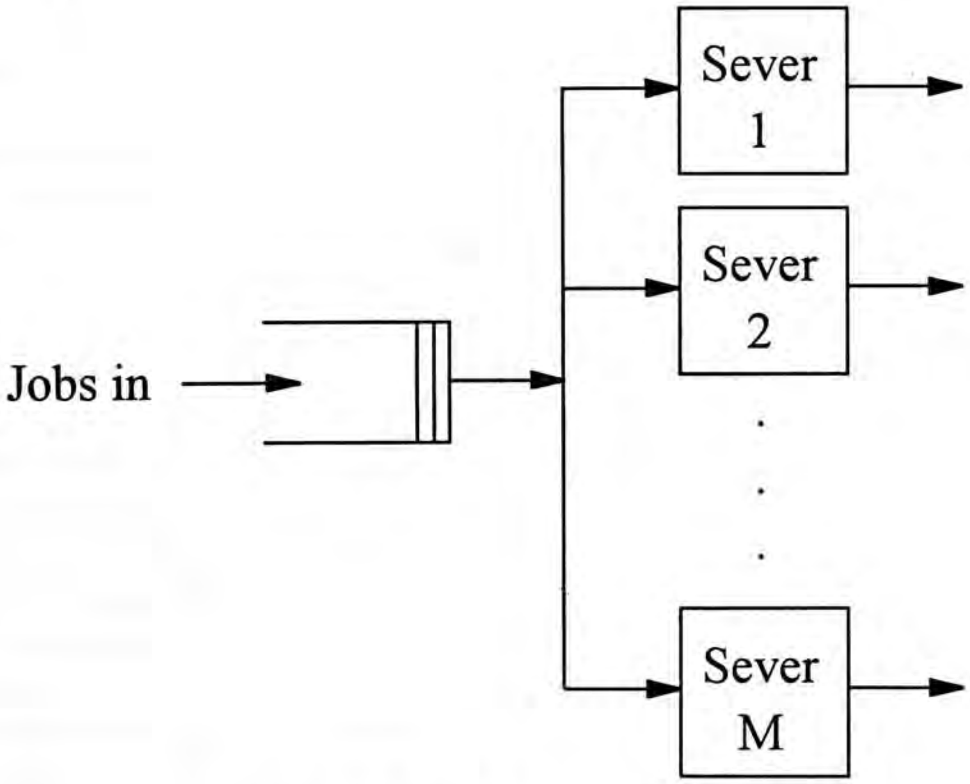


Figure 3-4 Queueing model of DMP Disk Array.

Let random variables N_w and N_r denote the number of "write" jobs and "read" jobs in the disk array, and N_q be the number of queueing jobs (including the HOL job) at any time. It is easy to see that the triplet (N_w,N_r,N_q) completely specify the state of the system. Let $S_{w,r,q}$ denotes the state of the system when $N_w=w$, $N_r=r$ and $N_q=q$. State transition will take place when a new job arrives or when a job in the system departs. Since the time spent in a state is exponentially distributed, the evolution of (N_w,N_r,N_q) is a continuous time Markov process. We define the transition probabilities to be

$$P[S_{w',r',q'}|S_{w,r,q}] = P\Big[(N_w, N_r, N_q) = (w',r',q') \text{ after state transition} \;\Big|\; (N_w, N_r, N_q) = (w,r,q) \text{ before state transition}\Big] \quad (3\text{-}11)$$

Consider a particular state transition at time *t*. Define events E_1, E_2, and E_3 as:

E_1: A new job arrives at time *t*.

E_2: A "read" job departs at time *t*.

E_3: A "write" job departs at time *t*.

These events are listed in the first column of Table 3-1. The disk array is at $S_{w,r,q}$ immediately before *t*, i.e. at time $t\text{-}\delta t$ ($\delta t \to 0$). The probability that a new job will arrive in the interval $(t\text{-}\delta t, t)$ is $\lambda\delta t$ if $\delta t \to 0$. Similarly, the probabilities that the disk array will finish serving a "write" job and a "read" job in this small time interval are $\mu_w\delta t$ and $\mu_r\delta t$ respectively. Therefore,

$$P[E_1] = \frac{\lambda}{\lambda + w\mu_w + r\mu_r}$$
$$P[E_2] = \frac{r\mu_r}{\lambda + w\mu_w + r\mu_r} \quad (3\text{-}12)$$
$$P[E_3] = \frac{w\mu_w}{\lambda + w\mu_w + r\mu_r}$$

When a new job arrives (i.e. E_1 occurs), the probabilities that this HOL job is of the "read" type and of the "write" type are $1\text{-}\alpha$ and α respectively. However different probability for each of the job type for the HOL job is found when a job departs (i.e. either E_2 or E_3 occurs). We denote the probabilities that the HOL job is of the "read" type and of the "write" type by h_w and $1\text{-}h_w$ respectively for system transitions due to job departures. Probability h_w can be derived as follows. At the previous state change, the HOL job was blocked because it requires the access of one or more busy disks. By that time there were $2w+r$ busy disks. If the HOL job is of the "read" type, the probability of blocking k_r is:

$$k_r = \frac{\text{number of busy disks}}{\text{total number of disks}} = \frac{2w+r}{M} \quad (3\text{-}13)$$

On the other hand, if the HOL job is of the "write" type, the probability of blocking k_w can be derived as follows. We shall call the disk which the HOL job targets for data modification the *data disk*, and the R disks storing the required parity information the *parity disks*. Let events ξ_1 and ξ_2 be:

ξ_1: The data disk to be accessed was free at the previous state change.

ξ_2: At least one of the *R* parity disks was free at the previous state change.

Then,

$$\begin{aligned} 1 - k_w &= P[\text{no blocking}] \\ &= P[\xi_1 \text{ } and \text{ } \xi_2] \\ &= P[\xi_1]P[\xi_2|\xi_1] \end{aligned} \quad (3\text{-}14)$$

$P[\xi_2 | \xi_1]$ is given by:

$$P[\xi_2|\xi_1]=1-\frac{\begin{pmatrix}\text{number of ways to}\\ \text{choose } R \text{ parity disks}\\ \text{from } 2w+r \text{ busy disks}\end{pmatrix}}{\begin{pmatrix}\text{number of ways to}\\ \text{choose R parity disks}\\ \text{from } M\text{-}1 \text{ disks}\end{pmatrix}}=\begin{cases}1 & 2w+r<R\\ 1-\dfrac{\binom{2w+r}{R}}{\binom{M\text{-}1}{R}} & 2w+r\geq R\end{cases} \tag{3-15}$$

Substituting into (3-14) and solving for k_w we obtain

$$k_w=\begin{cases}\dfrac{2w+r}{M} & 2w+r<R\\ \dfrac{2w+r}{M}+\dfrac{M-(2w+r)}{M}\dfrac{\binom{2w+r}{R}}{\binom{M-1}{R}} & 2w+r\geq R\end{cases} \tag{3-16}$$

Having obtained k_r and k_w, h_w is given by:

$$h_w=\frac{k_w}{k_r+k_w} \tag{3-17}$$

Conditioning on event E_i and giving the type of the HOL job, the probabilities of having different numbers of jobs located in the queue and in the disk array are listed in the sixth column of Table 3-1. For example, the first row of Table 3-1 corresponds to the case that a "read" job enters an empty queue is blocked, or the number of "read" jobs and "write" jobs in the disk array remain the same and q' becomes 1. The probability of having the new triplet $(w',r',q')=(w,r,1)$ after t under the two given conditions is denoted as a_1. Similarly the fifth row of Table 3-1 corresponds to the case that a "write" job enters an empty queue and gets served immediately. The probability of having $(w',r',q')=(w+1,r,0)$ after t under the two given conditions is denoted by a_3 as shown. The derivation of all a_i's are given in the appendix.

The seventh column of Table 3-1 shows the type of the new HOL job and its corresponding probability. A new HOL job is blocked at time t with probabilities k'_r and k'_w when it is of the "read" type and of the "write" type respectively. Similar to the derivations on k_r and k_w, k'_r and k'_w can be obtained from (3-13) and (3-16) by changing the number of busy disks from $2w+r$ to $2w'+r'$.

The last column of Table 3-1 shows $P[S_{w',r',q'} \mid S_{w,r,q}, E_i]$. By removing the condition on E_i, the transition probabilities are thus obtained:

$$P[S_{w',r',q'}|S_{w,r,q}]=\sum_{i=1}^{3}P[S_{w',r',q'}|S_{w,r,q},E_i]P[E_i] \tag{3-18}$$

Having obtained the transition probabilities, the equilibrium distribution of different states can be computed in the usual way. The expected numbers of jobs in the queue are given by:

$$E[N_q]=\sum_{w=0}^{\lfloor M/2\rfloor}\sum_{r=0}^{M}\sum_{q=0}^{\infty}qP[S_{w,r,q}] \tag{3-19}$$

Finally, by Little's formula, the job's sojourn time D is given by

$$D = \frac{E[N_q]}{\lambda} + \left(\frac{\alpha}{\mu_w} + \frac{(1-\alpha)}{\mu_r} \right) \tag{3-20}$$

E_i	New states (w',r',q')			Type of the H.O.L job	$P[w',r',q' \mid E_i]$	Type of the new HOL job	$P[S_{w,r,q}, S_{w',r',q'} \mid E_i]$
E_1 {a new job arrives}	$q=0$	$q'=1$	$w'=w$, $r'=r$	read $(1-\alpha)$	a_1	-	$(1-\alpha)a_1$
				write (α)	a_2	-	αa_2
			all other new states	-	0	-	0
		$q'=0$	$w'=w$, $r'=r+1$	read $(1-\alpha)$	a_3	-	$(1-\alpha)a_3$
			$w'=w+1$, $r'=r$	write (α)	a_4	-	αa_4
			all other new states	-	0	-	0
		$q'>1$	all new states	-	0	-	0
	$q\geq1$	$q'=q+1$	$w'=w$, $r'=r$	-	1	-	1
			all other new states	-	0	-	0
		$q'\neq q+1$	all new states	-	0	-	0
E_2 {a read job departs}	$q=0$	$q'=0$	$w'=w$, $r'=r-1$	-	1	-	1
			all other new states	-	0	-	0
		$q'\neq0$	all new states	-	0	-	0
	$q=1$	$q'=1$	$w'=w$, $r'=r-1$	read $(1-h_w)$	a_5	-	$(1-h_w)a_5$
				write (h_w)	a_6	-	$h_w a_6$
			all other new states	-	0	-	0
		$q'>1$	all new states	-	0	-	0
		$q'=0$	$w'=w$, $r'=r$	read $(1-h_w)$	a_7	-	$(1-h_w)a_7$
			$w'=w+1$, $r'=r-1$	write (h_w)	a_8	-	$h_w a_8$
			all other new states	-	0	-	0
	$q>1$	$q'=q$	$w'=w$, $r'=r-1$	read $(1-h_w)$	a_5	-	$(1-h_w)a_5$
				write (h_w)	a_6	-	$h_w a_6$
			all other new states	-	0	-	0
		$q'>q$	all new states	-	0	-	0

Table 3-1 Transition probabilities of each event E_i for average delay analysis.

E_i	New states (w',r',q')			Type of the H.O.L job	$P[w',r',q' \mid E_i]$	Type of the new HOL job	$P[S_{w,r,q}, S_{w',r',q'} \mid E_i]$
E_2 (cont.)	$q>1$	$q'=q-1$	$w'=w$, $r'=r$	read $(1-h_w)$	a_7	read $((1-\alpha)k'_r)$	$(1-h_w)(1-\alpha)a_7k'_r$
						write $(\alpha k'_w)$	$(1-h_w)\alpha a_7k'_w$
			$w'=w+1$, $r'=r-1$	write (h_w)	a_8	read $((1-\alpha)k'_r)$	$h_w(1-\alpha)a_8k'_r$
						write $(\alpha k'_w)$	$h_w\alpha a_8k'_w$
			all other new states	-	0	-	0
		$q'<q-1$, $q'=0$	$w'=w$, $r'=r-1+(q-q')$	read $(1-h_w)$	a_9	-	$(1-h_w)a_9$
			$w'=w+(q-q')$, $r'=r-1$	write (h_w)	a_{10}	-	h_wa_{10}
			$w'\geq w+1$, $r'\geq r$, $(w'-w)+(r'-r+1)=q-q'$	read $(1-h_w)$	a_{11}	-	$(1-h_w)a_{11}$
				write (h_w)	a_{12}	-	h_wa_{12}
			all other new states	-	0	-	0
		$q'<q-1$, $q'>0$	$w'=w$, $r'=r-1+(q-q')$	read $(1-h_w)$	a_9	read $((1-\alpha)k'_r)$	$(1-h_w)(1-\alpha)a_9k'_r$
						write $(\alpha k'_w)$	$(1-h_w)\alpha a_9k'_w$
			$w'=w+(q-q')$, $r'=r-1$	write (h_w)	a_{10}	read $((1-\alpha)k'_r)$	$h_w(1-\alpha)a_{10}k'_r$
						write $(\alpha k'_w)$	$h_w\alpha a_{10}k'_w$
			$w'\geq w+1$, $r'\geq r$, $(w'-w)+(r'-r+1)=q-q'$	read $(1-h_w)$	a_{11}	read $((1-\alpha)k'_r)$	$(1-h_w)(1-\alpha)a_{11}k'_r$
						write $(\alpha k'_w)$	$(1-h_w)\alpha a_{11}k'_w$
				write (h_w)	a_{12}	read $((1-\alpha)k'_r)$	$h_w(1-\alpha)a_{12}k'_r$
						write $(\alpha k'_w)$	$h_w\alpha a_{12}k'_w$
			all other new states	-	0	-	0

Table 3-1 Transition probabilities of each event E_i for average delay analysis (cont'd).

E_i	New states (w',r',q')			Type of the H.O.L job	$P[w',r',q' \mid E_i]$	Type of the new HOL job	$P[S_{w,r,q}, S_{w',r',q'} \mid E_i]$
E_3 {a write job departs}	$q=0$	$q'=0$	$w'=w-1$, $r'=r$	-	1	-	1
			all other new states	-	0	-	0
		$q'\neq 0$	all new states	-	0	-	0
	$q=1$	$q'=1$	$w'=w-1$, $r'=r$	read $(1-h_w)$	a_{13}	-	$(1-h_w)a_{13}$
				write (h_w)	a_{14}	-	$h_w a_{14}$
			all other new states	-	0	-	0
		$q'>1$	all new states	-	0	-	0
		$q'=0$	$w'=w-1$, $r'=r+1$	read $(1-h_w)$	a_{15}	-	$(1-h_w)a_{15}$
			$w'=w$, $r'=r$	write (h_w)	a_{16}	-	$h_w a_{16}$
			all other new states	-	0	-	0
	$q>1$	$q'=q$	$w'=w-1$, $r'=r$	read $(1-h_w)$	a_{13}	-	$(1-h_w)a_{13}$
				write (h_w)	a_{14}	-	$h_w a_{14}$
			all other new states	-	0	-	0
		$q'>q$	all new states	-	0	-	0
		$q'=q-1$	$w'=w-1$, $r'=r+1$	read $(1-h_w)$	a_{15}	read $((1-\alpha)k'_r)$	$(1-h_w)(1-\alpha)\, a_{15}k'_r$
						write $(\alpha k'_w)$	$(1-h_w)\alpha a_{15}k'_w$
			$w'=w$, $r'=r$	write (h_w)	a_{16}	read $((1-\alpha)k'_r)$	$h_w(1-\alpha)\, a_{16}k'_r$
						write $(\alpha k'_w)$	$h_w \alpha a_{16}k'_w$
			all other new states	-	0	-	0
		$q'<q-1$, $q'=0$	$w'=w-1$, $r'=r+(q-q')$	read $(1-h_w)$	a_{17}	-	$(1-h_w)a_{17}$
			$w'=w-1+(q-q')$, $r'=r$	write (h_w)	a_{18}	-	$h_w a_{18}$
			$w'\geq w$, $r'\geq r+1$,	read $(1-h_w)$	a_{19}	-	$(1-h_w)a_{19}$
			$(w'-w+1)+(r'-r)=q-q'$	write (h_w)	a_{20}	-	$h_w a_{20}$
			all other new states	-	0	-	0

Table 3-1 Transition probabilities of each event E_i for average delay analysis (cont'd).

E_i	New states (w',r',q')			Type of the H.O.L job	P[w',r',q' $\vert E_i$]	Type of the new HOL job	$P[S_{w,r,q}, S_{w',r',q'} \vert E_i]$
E_3 (cont.)	q>1	q'<q-1 q'>0	w'=w-1, r'=r+(q-q')	read (1-h_w)	a_{17}	read ((1-α)k'_r)	(1-h_w)(1-α) $a_{17}k'_r$
						write ($\alpha k'_w$)	(1-h_w)$\alpha a_{17}k'_w$
			w'=w-1+(q-q'), r'=r	write (h_w)	a_{18}	read ((1-α)k'_r)	h_w(1-α)$a_{18}k'_r$
						write ($\alpha k'_w$)	$h_w\alpha a_{18}k'_w$
			w'≥w, r'≥r+1, (w'-w+1)+(r'-r)=q-q'	read (1-h_w)	a_{19}	read ((1-α)k'_r)	(1-h_w)(1-α) $a_{19}k'_r$
						write ($\alpha k'_w$)	(1-h_w)$\alpha a_{19}k'_w$
				write (h_w)	a_{20}	read ((1-α)k'_r)	h_w(1-α)$a_{20}k'_r$
						write ($\alpha k'_w$)	$h_w\alpha a_{20}k'_w$
			all other new states	-	0	-	0

Table 3-1 Transition probabilities of each event Ei for average delay analysis (cont'd).

B) Numerical Example

As an example consider a small DMP Disk Array with data storage capacity of 4 disks. We assume in this example that μ_w=30 jobs/sec and μ_r=50 jobs/sec. Figures 5 to 7 shows both the analytic and simulation results for this disk array, and we observe that they match very well with each other. Note that for all simulation results shown in this chapter we have extended the simulation time sufficiently long to make the 95% confidence intervals smaller than the size of the simulation points shown.

Figure 3-5 shows the job delay against the arrival rate when all jobs are of "write" type. As indicated by the curve, the maximum throughput for RAID level 5 is about 32.5 jobs/sec. When one parity disk is added to the disk array (i.e. when M=6 & R=2), we find that the average job delay is reduced under all traffic conditions and the maximum throughput is increased by about 24% when compared to RAID level 5. If one more parity disk is used (R=3), we find that the job delay is further reduced under all traffic conditions and 40% increase in maximum throughput is observed.

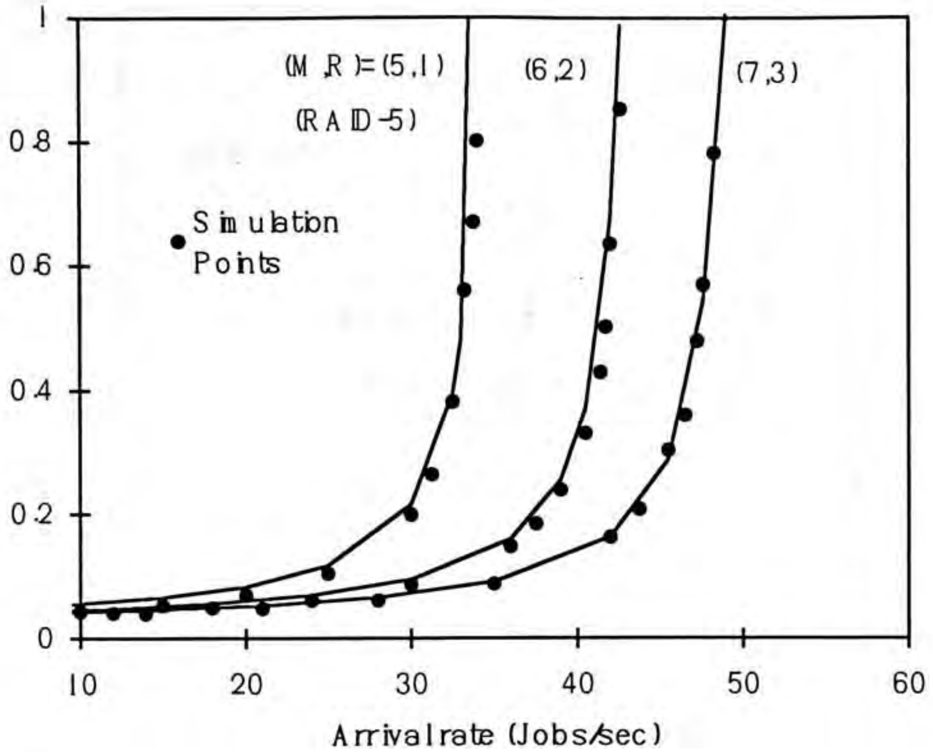


Figure 3-5 Average job delay against the arrival rate for small DMP Disk Arrays (data storage capacity M-R is 4 disks). All jobs are of "write" type ($\alpha=1$).

Figure 3-6 shows the job delay against the arrival rate when half the jobs are of the "write" type (α=0.5). We observe from the figure that DMP Disk Array with R=2 and R=3 again performs better than RAID level 5 under all traffic levels. Compared with RAID level 5, DMP Disk Array provides 13% and 23% increase in maximum throughput when R=2 and R=3 respectively. When all the jobs are of the "read" type (Figure 3-7), DMP Disk Array provides relatively small increase in throughput.

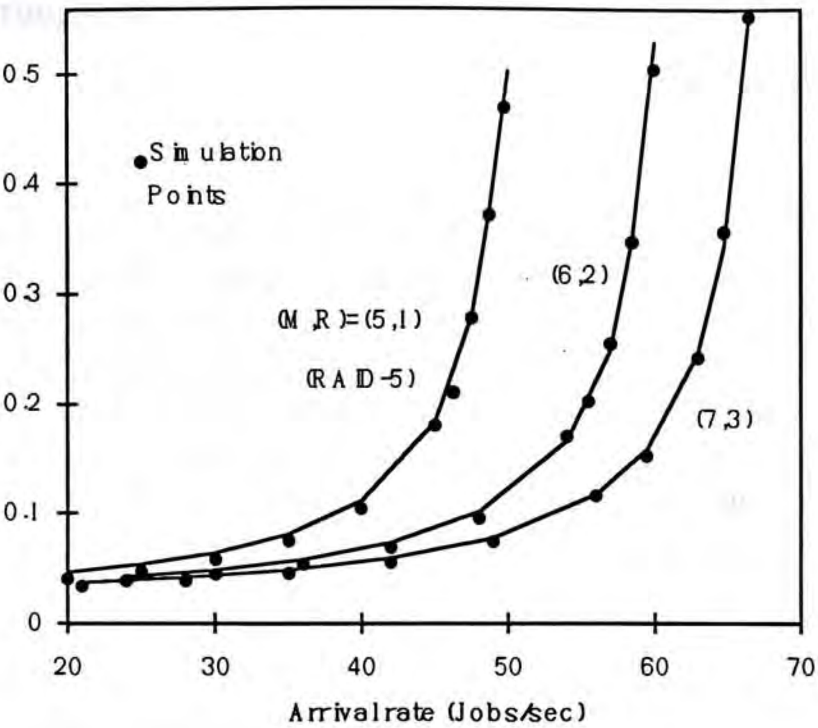


Figure 3-6 Average job delay against the arrival rate for small DMP Disk Arrays (data storage capacity M-R is 4 disks). Half the jobs are of "write" type (α=0.5).

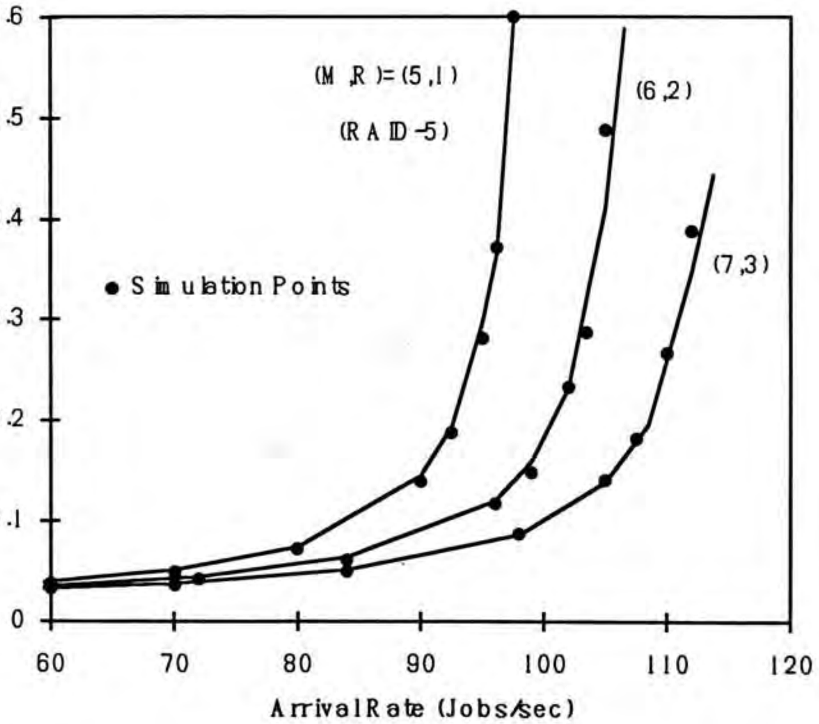


Figure 3-7 Average job delay against the arrival rate for small DMP Disk Arrays (data storage capacity M-R is 4 disks). All jobs are of "read" type (α=0).

3.4 Maximum Throughput

A) Analysis

Although the analysis given in section III provides an exact solution on average job delay, the computation is very demanding when the disk array is large. In the following, we present a simplified analysis which gives the maximum throughput for DMP Disk Array. A modified model shown in Figure 3-8 is used in our analysis. Compared with the previous model (Figure 3-4), the FCFS queue is removed and we assume that there is always a new job available at the input of the disk array. All other previously used assumptions are used in this maximum throughput analysis. Since the queue is removed, system's state can solely be specified by N_w and N_r,and is denoted by $S_{w,r}$. State transition will take place when a job in the disk array departs, i.e. either E_2 or E_3 occurs. Since the time spent in a state is exponentially distributed, the evolution of N_w and N_r remains to be a continuous time Markov process. As before, we define the transition probabilities to be

$$P[S_{w,r}, S_{w',r'}] =$$

$$P[N_w = w', N_r = r' \text{ after system transition } | N_w = w, N_r = r \text{ before system transition}] \quad (3\text{-}21)$$

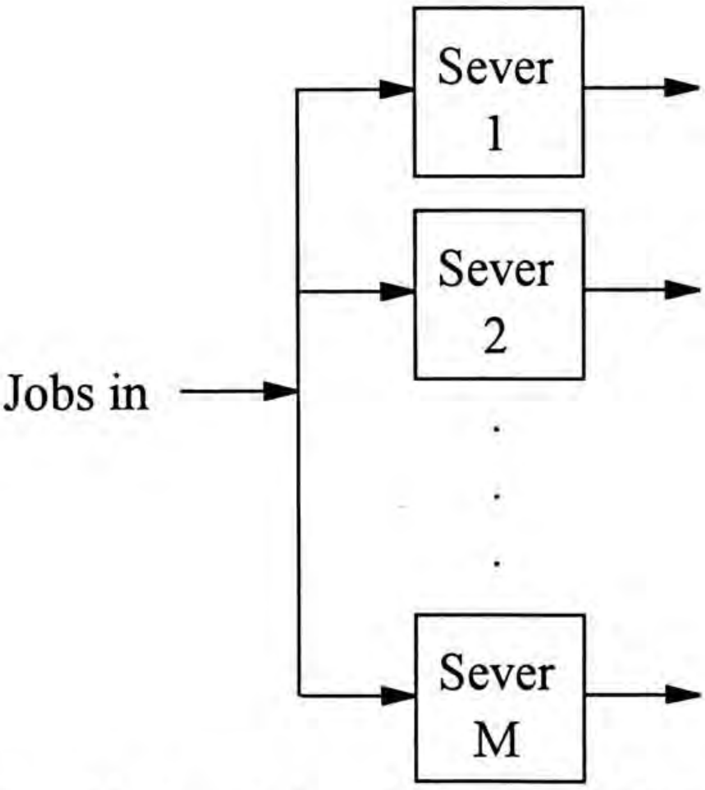


Figure 3-8 The modified queueing model for DMP Disk Array.

Consider a particular state transition occurs at time t. The probability of occurrence for E_2 and E_3 are given by

$$P[E_2] = \frac{r\mu_r}{w\mu_w + r\mu_r}$$

$$P[E_3] = \frac{w\mu_w}{w\mu_w + r\mu_r} \tag{3-22}$$

Given that a specific event E_i occurs, the transition probabilities $P[S_{w,r}, S_{w',r'} | E_i]$ are listed in the last column of Table 3-2. The derivations on probabilities a_i are given in the appendix. Having obtained the transition probabilities, the equilibrium distribution of different states can be computed as before. The expected time between successive job departures X is given by

$$X = \sum_{w=0}^{\lfloor M/2 \rfloor} \sum_{r=0}^{M} P[S_{w,r}] \frac{1}{w\mu_w + r\mu_r} \tag{3-23}$$

Finally, the maximum throughput for DMP Disk Arrays T is

$$T = \frac{1}{X} \tag{3-24}$$

E_i	New States (w',r')	Type of the H.O.L job	$P[w',r' \mid E_i]$	Type of the new HOL job	$P[S_{w,r}, S_{w',r'} \mid E_i]$
E_2 {a read job departs}	$w'=w, r'=r-1$	read $(1-h_w)$	a_5	-	$(1-h_w)a_5$
		write (h_w)	a_6	-	h_wa_6
	$w'=w, r'=r$	read $(1-h_w)$	a_7	read $((1-\alpha)k'_r)$	$(1-h_w)(1-\alpha)a_7k'_r$
				write $(\alpha k'_w)$	$(1-h_w)\alpha a_7k'_w$
	$w'=w+1, r'=r-1$	write (h_w)	a_8	read $((1-\alpha)k'_r)$	$h_w(1-\alpha)a_8k'_r$
				write $(\alpha k'_w)$	$h_w\alpha a_8k'_w$
	$w'=w, r'>r$	read $(1-h_w)$	a_9	read $((1-\alpha)k'_r)$	$(1-h_w)(1-\alpha)a_9k'_r$
				write $(\alpha k'_w)$	$(1-h_w)\alpha a_9k'_w$
	$w'>w+1, r'=r-1$	write (h_w)	a_{10}	read $((1-\alpha)k'_r)$	$h_w(1-\alpha)a_{10}k'_r$
				write $(\alpha k'_w)$	$h_w\alpha a_{10}k'_w$
	$w'\geq w+1, r'\geq r$	read $(1-h_w)$	a_{11}	read $((1-\alpha)k'_r)$	$(1-h_w)(1-\alpha)a_{11}k'_r$
				write $(\alpha k'_w)$	$(1-h_w)\alpha a_{11}k'_w$
		write (h_w)	a_{12}	read $((1-\alpha)k'_r)$	$h_w(1-\alpha)a_{12}k'_r$
				write $(\alpha k'_w)$	$h_w\alpha a_{12}k'_w$
	all other new states	-	0	-	0

Table 3-2 Transition probabilities of each event E_i for maximum throughput analysis.

E_i	New States (w',r')	Type of the H.O.L job	$P[w',r' \mid E_i]$	Type of the new HOL job	$P[S_{w,r}, S_{w',r'} \mid E_i]$
E_3 {a write job departs}	$w'=w-1$, $r'=r$	read $(1-h_w)$	a_{13}	-	$(1-h_w)a_{13}$
		write (h_w)	a_{14}	-	$h_w a_{14}$
	$w'=w-1$, $r'=r+1$	read $(1-h_w)$	a_{15}	read $((1-\alpha)k'_r)$	$(1-h_w)(1-\alpha)a_{15}k'_r$
				write $(\alpha k'_w)$	$(1-h_w)\alpha a_{15}k'_w$
	$w'=w$, $r'=r$	write (h_w)	a_{16}	read $((1-\alpha)k'_r)$	$h_w(1-\alpha)a_{16}k'_r$
				write $(\alpha k'_w)$	$h_w\alpha a_{16}k'_w$
	$w'=w-1$, $r'>r+1$	read $(1-h_w)$	a_{17}	read $((1-\alpha)k'_r)$	$(1-h_w)(1-\alpha)a_{17}k'_r$
				write $(\alpha k'_w)$	$(1-h_w)\alpha a_{17}k'_w$
	$w'>w$, $r'=r$	write (h_w)	a_{18}	read $((1-\alpha)k'_r)$	$h_w(1-\alpha)a_{18}k'_r$
				write $(\alpha k'_w)$	$h_w\alpha a_{18}k'_w$
	$w'\geq w$, $r'\geq r+1$	read $(1-h_w)$	a_{19}	read $((1-\alpha)k'_r)$	$(1-h_w)(1-\alpha)a_{19}k'_r$
				write $(\alpha k'_w)$	$(1-h_w)\alpha a_{19}k'_w$
		write (h_w)	a_{20}	read $((1-\alpha)k'_r)$	$h_w(1-\alpha)a_{20}k'_r$
				write $(\alpha k'_w)$	$h_w\alpha a_{20}k'_w$
	all other new states	-	0	-	0

Table 3-2 Transition probabilities of each event E_i for maximum throughput analysis (cont'd).

B) Numerical Examples

Figure 3-9 shows the throughput gain over that of RAID level 5 against *R* for a small DMP Disk Array with data storage capacity of 4 disks. We observe from the figure that increasing the number of parity disks *R* will always increase the maximum throughput for DMP Disk Array. When the proportion of "write" jobs is higher, the increase in maximum throughput is more apparent because DMP Disk Array will reduce the queueing time of "write" jobs. Considering the case of all "write" jobs ($\alpha=1$), DMP Disk Array with $R=2$ provides 24% increase in maximum throughput when compared to RAID level 5. Further increase *R* to 3 provides an additional 17% increase in maximum throughput. These performance figures match well with the numerical examples given in the previous section. When *R* is greater than 3, linear increase in maximum throughput is observed for each parity disk added.

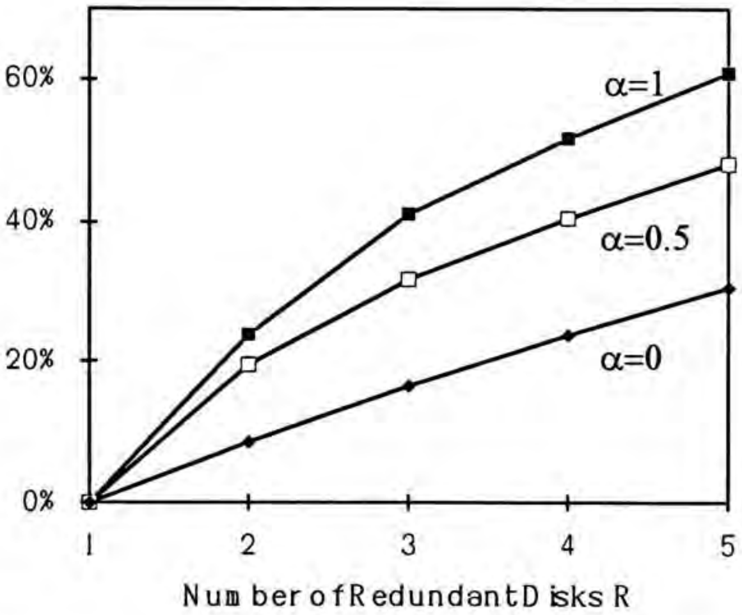


Figure 3-9 Throughput gain over that of RAID level 5 as a function of R for small DMP Disk Arrays (data storage capacity M-R is 4 disks).

Figure 3-10 plots the throughput gain over that of RAID level 5 against *R* for a large DMP Disk Array with data storage capacity of 24 disks. When compared with Figure 3-9, we observe that DMP Disk Array with $R=2$ provides even more notable increase in maximum throughput than the previous case. We can thus conclude that DMP Disk Array with $R=2$ provides the best cost/performance ratio.

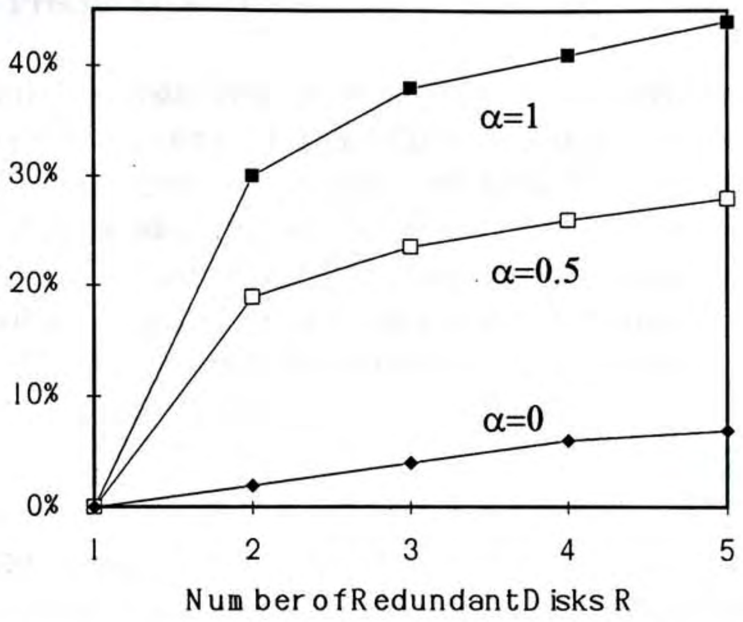


Figure 3-10 Throughput gain over that of RAID level 5 as a function of R for large DMP Disk Arrays (data storage capacity M-R is 24 disks).

3.5 Simulation with Precise Disk Model

In our previous analysis disk service time is assumed to be exponentially distributed. This is usually not true for practical disk drives. To better understand the performance of DMP Disk Array in practice, we perform simulation on DMP Disk Array with precise disk model. In our simulation, disks are *not* assumed to be rotationally synchronized and their simulation parameters are summarized in Table 3-3. Each disk access involves a seek time, a latency and a data transfer time. We use the seek profile in [16], which states that the seek time T_{seek} (in mSec) is related to seek distance x (in number of cylinders) by:

$$T_{seek} = \begin{cases} 0 & \text{for } x = 0 \\ 0.4623\sqrt{x-1} + 0.0092(x-1) + 2 & \text{for } x > 0 \end{cases} \qquad (3\text{-}25)$$

Latency is assumed to be uniformly distributed. Data transfer time for one sector is equal to the disk revolution time divided by the number of sectors per track as given in Table 3-3 With that, the mean service time for "write" jobs is computed to be 33.3 ms, and that for "read" jobs it is 20 ms. The corresponding service rates are therefore the same as those in the previous examples. As stated in [17], this kind of disk modeling provides more than 94% accuracy when ignoring the disk caching effect. Since disk caching has little impact on "write" performance (which we are most interested in), we can thus assume that the system has no disk caching mechanism.

Cylinders per disk	1024
Tracks per cylinder	14
Sectors per track	48
Bytes per sector	512
Revolution time	13.3 ms
Single cylinder seek time	2 ms
Average seek time	13 ms
Max. data transfer rate	1.7 MB/s

Table 3-3 Disk parameters used in simulation.

Figures 3-11 to 3-16 show the simulation results with precise disk model. We first consider a small disk array with data storage capacity of 4 disks and all jobs are of the "write" type. Figure 3-11 shows that the maximum throughput for RAID level 5 is about 37 jobs/sec. If DMP Disk Array with M=6 and R=2 is used, the average job delay is reduced under all traffic conditions and the maximum throughput is increased by 40% as compared to RAID level 5. This example shows that DMP Disk Array gives 40% increase of maximum throughput with only 20% increase of system cost (the increase of disk number from 5 to 6). If the cost of the disk controller is included, the increase of system cost will be even smaller. When the number of parity disk R is further increased to 3, the maximum throughput is 57% higher than that of RAID level 5.

Figure 3-12 shows the results for α=0.5 or half "read" and half "write" type of job mixture. We observe that DMP Disk Array with R=2 and R=3 provides 23% and

39% increase of I/O throughput respectively. When all the jobs are of "read" type, Figure 3-13 shows that DMP Disk Array provides relatively smaller throughput increase.

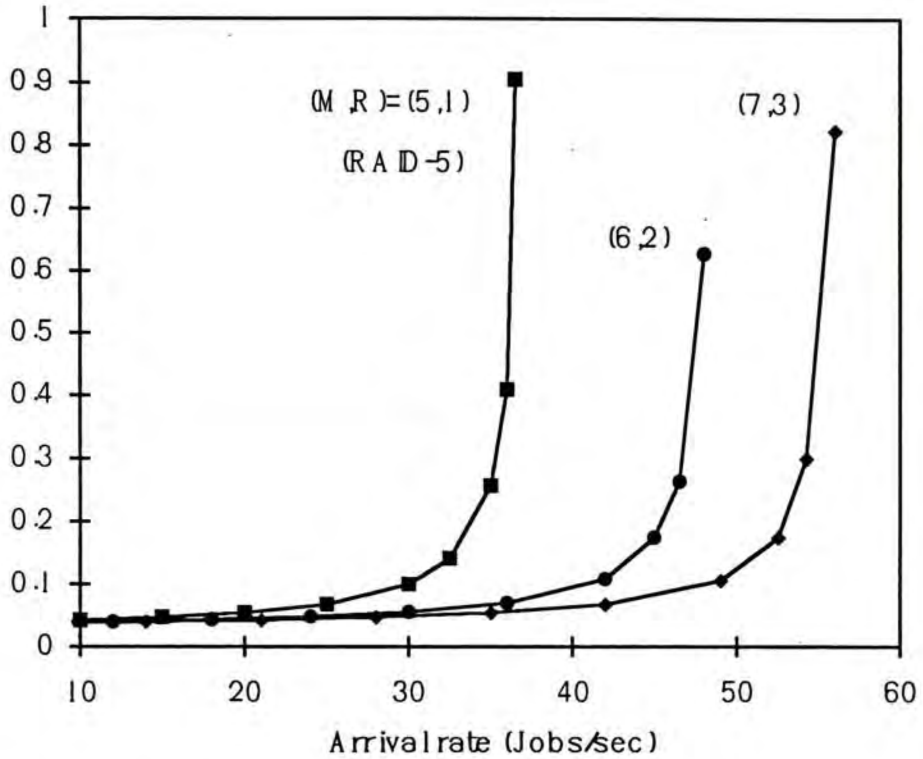


Figure 3-11 Simulation results on average job delay against arrival rate for small disk arrays (data storage capacity M-R is 4 disks). All jobs are of "write" type ($\alpha = 1$).

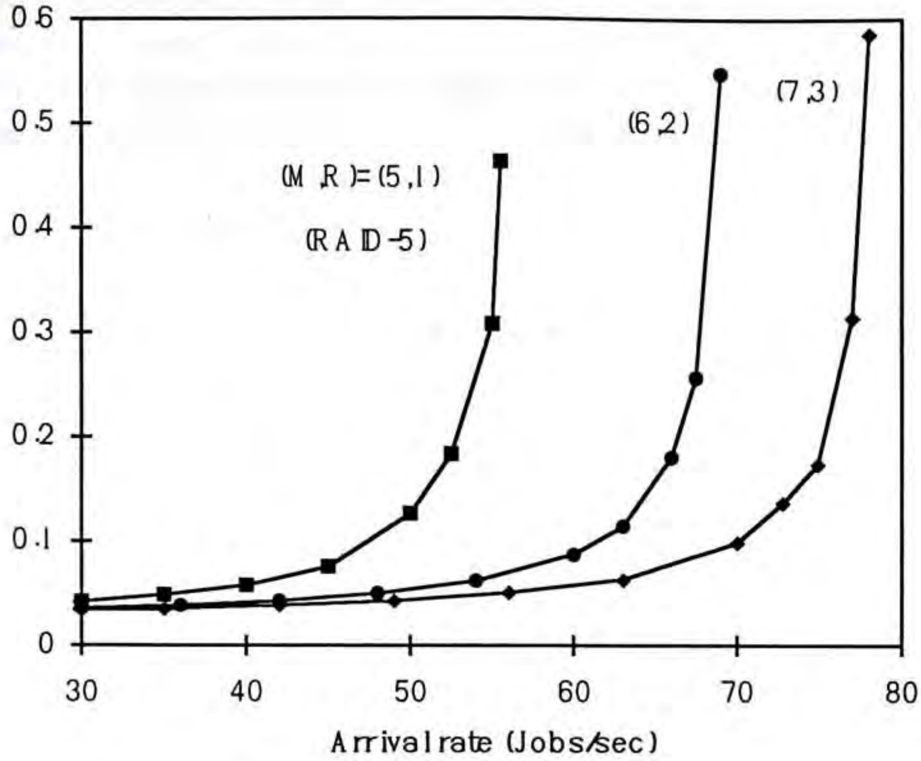


Figure 3-12 Simulation results on average job delay against arrival rate for small disk arrays (data storage capacity M-R is 4 disks). Half the jobs are "write" (α=0.5).

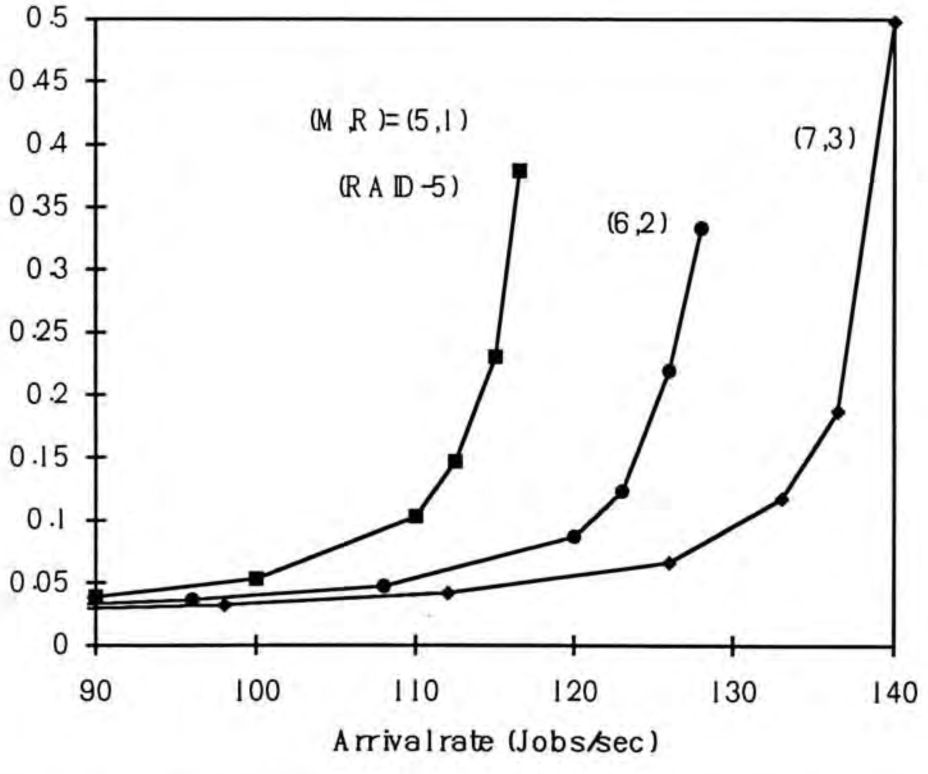


Figure 3-13 Simulation results on average job delay against arrival rate for small disk arrays (data storage capacity M-R is 4 disks). All jobs are of "read" type (α=0).

Figures 3-14 to 3-16 show the delay throughput characteristics of a typical large disk arrays with data storage capacity of 24 disks. DMP Disk Array again provides significant I/O throughput increase. For the case of all "write" jobs (Figure 3-14), the increase on maximum throughput for DMP Disk Array with $R=2$ is about 32% whereas the corresponding increase in system cost is at most 4%. Under the condition of equal number of "read" and "write" jobs (Figure 3-15), we find that the 4% increase in system cost can still give 20% higher throughput. Figure 3-16 shows that DMP Disk Array only provides minimal throughput increase when all the jobs are of the "read" type.

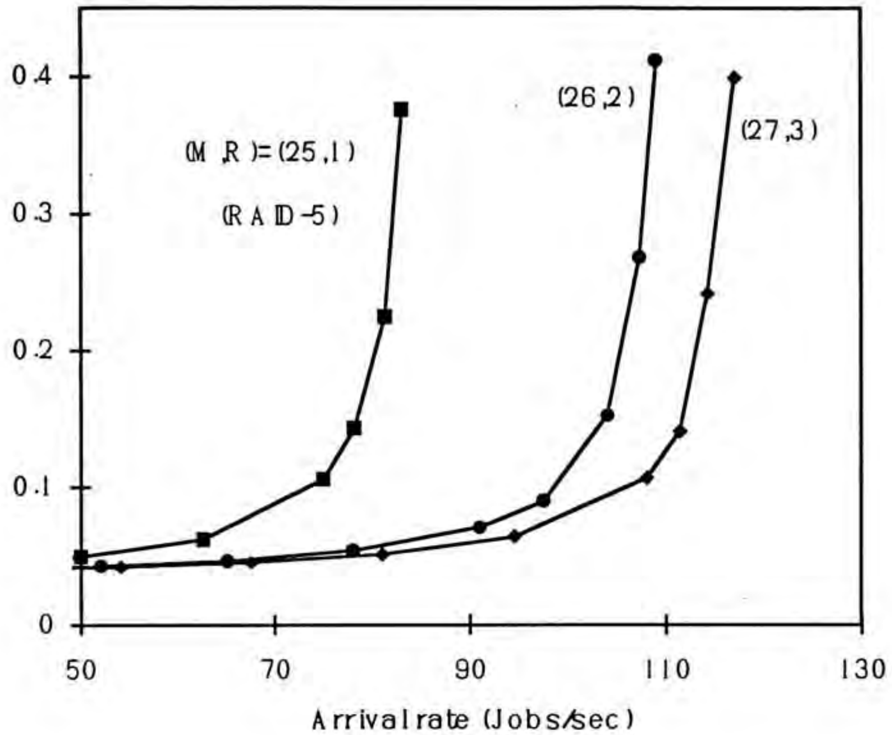


Figure 3-14 Simulation results on average job delay against arrival rate for large disk arrays (data storage capacity M-R is 24 disks). All jobs are of "write" type ($\alpha=1$).

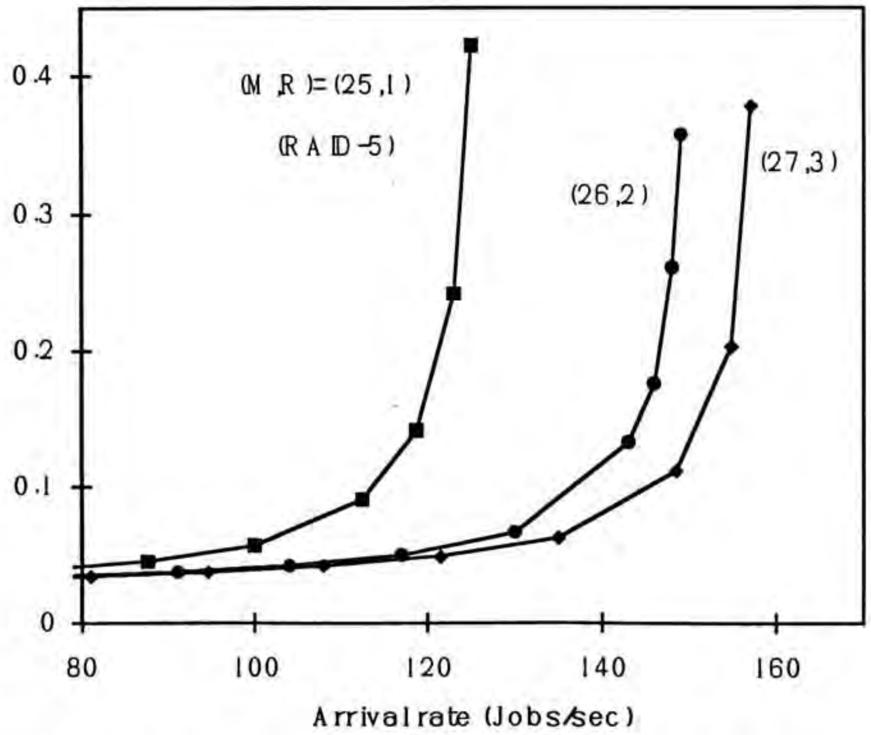


Figure 3-15 Simulation results on average job delay against arrival rate for large disk arrays (data storage capacity M-R is 24 disks). Half the jobs are of "write" type ($\alpha=0.5$).

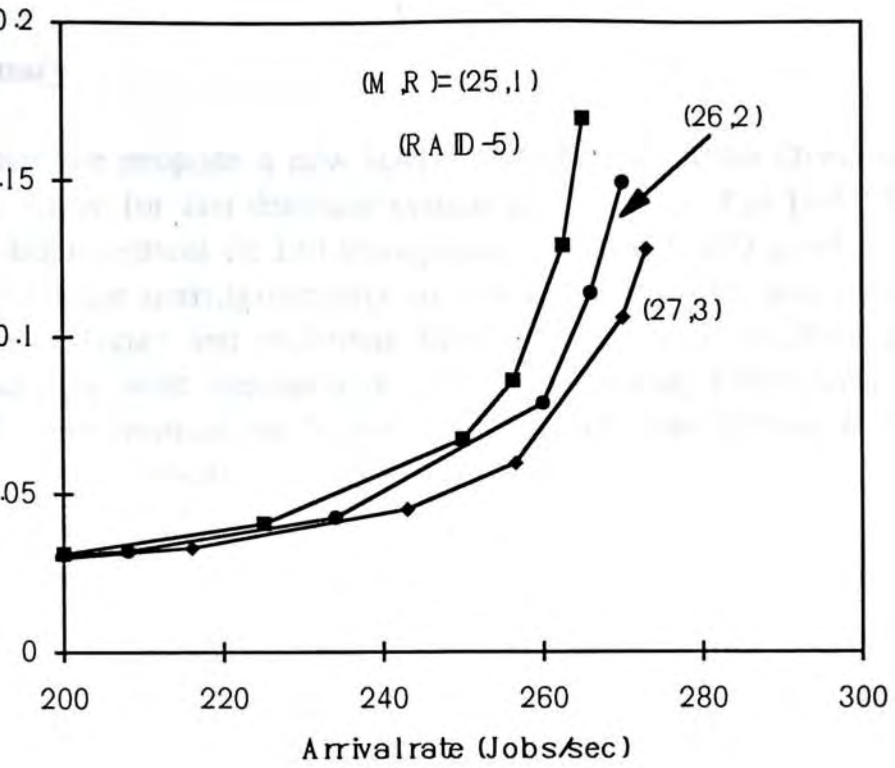


Figure 3-16 Simulation results on average job delay against arrival rate for large disk arrays (data storage capacity M-R is 24 disks). All jobs are of "read" type ($\alpha=0$).

3.6 Chapter Summary

In this chapter we propose a new RAID architecture called Dynamic Multiple Parity (DMP) Disk Array for fast database system applications. The DMP Disk Array provide significant improvement on I/O throughput over the RAID level 5. The DMP Disk Array also inherit the sparing property so that it has a higher survivability under disk failure conditions. Delay and maximum throughput analysis on DMP Disk Array is performed. Simulation with precise disk model shows that DMP Disk Array can provide 30 to 40% improvement on "write" performance over that of RAID level 5 when one extra parity disk is used.

References

[1] Stanley Y W Su, *Database Computers*, McGraw-Hill Book Company, 1988.

[2] Gary Herman, Gita Gopal, K C Lee & Abel Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems," *Proc. ACM SIGMOD*, May 1987.

[3] Tom Bowen, Gita Gopal, Gary Herman and William Mansfield, "A Scale Database Architecture for Network Services," *IEEE Communications Magazine*, Vol.29, No.1, January 1991.

[4] D A Patterson, G A Gibson, and R H Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *ACM SIGMOD*, Chicago, 1988).

[5] R H Katz, *Information Management for Engineering Design*, Springer-Verlag, 1985.

[6] J L Encarnacao and P C Lockemann, *Engineering Databases*, Springer-Verlag, 1990.

[7] M. Singhal and N. G. Shivaratri, *Advanced Concepts in Operating Systems*, chapter 20, McGraw-Hill, 1994.

[8] S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*, chapter 8, McGraw-Hill, 1984.

[9] P. A. Bernstein, D. W. Shipman, and J. B. Rothnie, "Concurrency Control in a System for Distributed Databases (SDD-1)," *ACM Transactions on Database Systems*, vol.5, no.1, 1980.

[10] E K Lee and R H Katz, "The Performance of Parity Placements in Disk Arrays," *IEEE Transactions on Computers*, vol.42, no.6, June 1993.

[11] J Menon and D Mattson, "Comparison of Sparing Alternatives for Disk Array," *Computer Architecture News*, Vol.20, Iss.2, May 1992.

[12] Michelley Y. Kim, "Synchronized Disk Interleaving," *IEEE Transactions on Computers*, vol.35, no.11, pp.978-988, November, 1986.

[13] Neil C. Wilhelm, "A General Model for the Performance of Disk Systems," *Journal of the ACM*, vol.24, no.1, pp.14-31, January, 1977.

[14] Spencer W. Ng, "Improving Disk Performance Via Latency Reduction," *IEEE*

Transactions on Computers, vol.40, no.1, pp.22-30, January, 1991.

[15] A. L. Narasimha Reddy and Prithviraj Banerjee, "An Evaluation of Multiple-Disk I/O Systems," *IEEE Transactions on Computers*, vol.38, no.12, December, 1989.

[16] E K Lee and R H Katz, "The Performance of Parity Placements in Disk Arrays," *IEEE Trans. on Computers*, Vol.42, No.6, pp.651-664, June 1993.

[17] C Ruemmler and J Wilkes, "An Introduction to Disk Drive Modeling," *IEEE Computer Magazine*, Vol.27, No.3, pp.17-28, March 1994.

Appendix

In this appendix, we derive a_1 to a_{20}.

A. Under condition E_1 (job arrivals)

There are four cases to consider.

Case 1: ***$q=0$, $q'=1$, $w'=w$, $r'=r$, a "read" job arrives***

In this case a "read" job arrives to an empty queue and is blocked. This occurs when the data disk is busy or with probability a_1 given by

$$a_1 = \frac{\text{number of busy disks}}{\text{total number of disks}} = \frac{2w+r}{M} \tag{3-A1}$$

Case 2: ***$q=0$, $q'=1$, $w'=w$, $r'=r$, a "write" job arrives***

In this case a "write" job arrives to an empty queue and is blocked. Similar to the derivations on k_w,the probability of this type of state change a_2 is

$$a_2 = \begin{cases} \dfrac{2w+r}{M} & 2w+r < R \\[2ex] \dfrac{2w+r}{M} + \dfrac{M-(2w+r)}{M} \dfrac{\binom{2w+r}{R}}{\binom{M-1}{R}} & 2w+r \geq R \end{cases} \tag{3-A2}$$

Case 3: ***$w'=w$, $r'=r+1$,*** **$q'=0$, $q=0$**

In this case a "read" job arrives and is served immediately. The probability of this type of state change is $a_3 = 1 - a_1$.

Case 4: ***$w'=w+1$, $r'=r$,*** **$q'=0$, $q=0$**

In this case a "write" job arrives and is served immediately. The probability of this type of state change is $a_4 = 1 - a_2$.

B. Under condition E_2 ("read" job departure)

There are eight cases to consider.

Case 1: $q \geq 1$, *$q'=q$, $w'=w$, $r'=r-1$*, HOL job is of the "read" type

In this case the HOL job is blocked when a "read" job departs. This occurs when the data disk is busy or with probability

$$a_5 = \frac{2w+r-1}{2w+r} \tag{3-A3}$$

Case 2: $q \geq 1$, $q'=q$, $w'=w$, $r'=r-1$, HOL job is of the "write" type

This is the same as case 1 except the HOL job is of the "write" type. Let $\xi_3(y)$ be the event that at least one of the parity disks is free given that y busy disks are busy. Similar to the derivations leading to (15), we have

$$P[\xi_3(y)] = \begin{cases} 1 & \text{for } y < R \\ 1 - \dfrac{\binom{y}{R}}{\binom{M-1}{R}} & \text{for } y \geq R \end{cases} \tag{3-A4}$$

Define ξ_4 and ξ_5 be the events:

- ξ_4: In the previous state change, the HOL job was blocked because the data disk is busy.
- ξ_5: In the previous state change, although the data disk is free the HOL job was blocked because all parity disks are busy.

When the HOL job was blocked in the previous state change, there were $2w+r$ busy disks. Therefore, similar to the derivations on k_r and k_w, P[ξ_4] and P[ξ_5] are given by:

$$P[\xi_4] = \frac{\left(\frac{2w+r}{M}\right)}{\left(\frac{2w+r}{M}\right) + \left(\frac{M-(2w+r)}{M}(1-P[\xi_3(2w+r))\right)} \tag{3-A5}$$

$$P[\xi_5] = \frac{\left(\frac{M-(2w+r)}{M}(1-P[\xi_3(2w+r))\right)}{\left(\frac{2w+r}{M}\right) + \left(\frac{M-(2w+r)}{M}(1-P[\xi_3(2w+r))\right)} \tag{3-A6}$$

Consider three sub-cases for this case:

Sub-case 2.1: $2w+r<R$

In this sub-case, the number of busy disks before state change is less than R. This implies that P[ξ_4]=1 and P[ξ_5] =0. By using the same argument in deriving a_5, the probability of this type of state change is the same as a_5.

Sub-case 2.2: $2w+r=R$

In this sub-case, both ξ_4 and ξ_5 are possible. Define ξ_6 and ξ_7 as the events:

- ξ_6: The HOL job blocked in the previous state change is blocked because the data disk is still busy.
- ξ_7: Although the data disk becomes free at the state change, the HOL job is still blocked because all parity disks are busy.

Given that ξ_4 occurred, by using the same argument in deriving a_5 P[$\xi_6 | \xi_4$] is given by:

$$P[\xi_6 | \xi_4] = \frac{2w+r-1}{2w+r} \tag{3-A7}$$

On the other hand, P[$\xi_7 | \xi_4$]=0 because the number of busy disks is less than R after the "read" job departs. If ξ_5 occurred in the previous state change, it means that all R busy disks were parity disks at that time. Therefore, both ξ_6 and ξ_7 are not possible to occur at the state change. The probability of this type of state change is thus:

$$a_6 = P[\xi_6|\xi_4]P[\xi_4] \tag{3-A8}$$

Sub-case 2.3 $2w+r>R$

This sub-case is the same as sub-case 2.2 with the exception that both P[ξ_7| ξ_4] and P[ξ_7| ξ_5] are not zeros. Given that ξ_4 occurred in the previous state change, ξ_7 happens if i) the data disk is freed at the state change; and ii) all parity disks are busy. P[ξ_7| ξ_4] is then given by:

$$P[\xi_7|\xi_4] = \frac{1}{2w+r}\frac{\binom{2w+r-1}{R}}{\binom{M-1}{R}} \tag{3-A9}$$

If ξ_5 occurred in the previous state change, it means that all parity disks were busy at that time. For ξ_7 to occur it is necessary that no parity disk is freed at the state change. Therefore, P[ξ_7| ξ_5] in this sub-case is given by:

$$P[\xi_7|\xi_5] = \frac{2w+r-R}{2w+r} \tag{3-A10}$$

The probability of this type of state change is:

$$a_6 = P[\xi_6|\xi_4]P[\xi_4] + P[\xi_7|\xi_4]P[\xi_4] + P[\xi_7|\xi_5]P[\xi_5] \tag{3-A11}$$

Case 3: *q≥1, q'=q-1, w'=w, r'=r*

In this case, the HOL job blocked in the pervious state change (which is of the "read" type) is served after the state change. This implies that the data disk is freed at the state change. The probability of this type of state change is therefore:

$$a_7 = 1 - a_5 = \frac{1}{2w+r} \tag{3-A12}$$

Case 4: *q≥1, q'=q-1, w'=w+1, r'=r-1*

This is the same as case 3 with the exception that the HOL job is of the "write" type. Therefore, the probability of this type of state change is a_8=1-a_6.

Case 5: *q>1, q'<q-1, w'=w, r'=r-1+(q-q')*

In this case, the HOL job is of the "read" type and q-q'-1 or r'-r read jobs (excluding the HOL job) are served . The probability that q-q'-1 successive jobs are all of the "read" type is (1-α)$^{q-q'-1}$. Similar to the derivations on k_r, the probability of this type of state change is

$$a_9 = P\begin{bmatrix}\text{The HOL}\\ \text{job is not}\\ \text{blocked}\end{bmatrix} P\begin{bmatrix}q\text{-}q'\text{-1 successive}\\ \text{jobs are all of the}\\ \text{"read" type}\end{bmatrix} P\begin{bmatrix}\text{The first}\\ \text{"read" job is}\\ \text{not blocked}\end{bmatrix} P\begin{bmatrix}\text{The second}\\ \text{"read" job is}\\ \text{not blocked}\end{bmatrix} \ldots P\begin{bmatrix}\text{The } q\text{-}q'\text{-1}^{\text{th}}\\ \text{"read" job is}\\ \text{not blocked}\end{bmatrix}$$

$$= a_7(1-\alpha)^{q-q'-1}\left(\frac{M-(2w+r)}{M}\right)\left(\frac{M-(2w+r)-1}{M}\right)\ldots\left(\frac{M-(2w+r)-\{(q-q'-1)-1\}}{M}\right)$$

(3-A13)

Case 6: ***q>1, q'<q-1, w'=w+(q-q'), r'=r-1***

In this case, the HOL job is of the "write" type and *q-q'-1* or *w'-w-1* "write" jobs are served. By using the same argument as in the previous case and in the derivations on k_w, the probability of this type of state change is:

$$a_{10} = P\begin{bmatrix}\text{The HOL}\\ \text{job is not}\\ \text{blocked}\end{bmatrix} P\begin{bmatrix}q\text{-}q'\text{-1 successive}\\ \text{jobs are all of the}\\ \text{"write" type}\end{bmatrix} P\begin{bmatrix}\text{The first}\\ \text{"write" job is}\\ \text{not blocked}\end{bmatrix} P\begin{bmatrix}\text{The second}\\ \text{"write" job is}\\ \text{not blocked}\end{bmatrix} \ldots P\begin{bmatrix}\text{The } q\text{-}q'\text{-1}^{\text{th}}\\ \text{"write" job is}\\ \text{not blocked}\end{bmatrix}$$

$$= a_8\alpha^{q-q'-1}\left(\frac{M-(2w+r)-1}{M}P[\xi_3((2w+r)+1)]\right)\left(\frac{M-(2w+r)-3}{M}P[\xi_3((2w+r)+3)]\right)$$
$$\ldots\left(\frac{M-(2w+r)-1-2\{(q-q'-1)-1\}}{M}P[\xi_3((2w+r)+1+2\{(q-q'-1)-1\})]\right)$$

(3-A14)

Case 7: ***q>1, q'<q-1, w'≥w+1, r'≥r, (w'-w)+(r'-(r-1))=q-q'***

In this case, the HOL job is of the "read" type and *q-q'*-1 new jobs (excluding the HOL jobs) which not all of them are of the same type are served. Let $n_w(i)$ be the number of "write" job being brought to the disk array by the i^{th} new job (excluding the HOL job), where i=1,2,3,....*q-q'*-1. Obviously, $n_w(i)=1$ if the i^{th} new job is of the "write" type and $n_w(i)=0$ otherwise. Since in this case *w'-w* "write" jobs are served, $\sum_{i=1}^{q-q'-1} n_w(i) = w'\text{-}w$. Let $n_b(i)$ be the number of busy disks in the disk array just *before* the i^{th} new job enters the disk array. Thus, $n_b(1)=2w+r$ and

$$n_b(i+1) = n_b(i) + (n_w(i)+1) \qquad \text{(3-A15)}$$

The probability that the i^{th} new job is not blocked, denoted as $p(i)$, is given by:

$$p(i) = \begin{cases} \dfrac{M-n_b(i)}{M} & \text{if the } i^{\text{th}} \text{ new job is of the "read" type, or } n_w(i)=0 \\ \dfrac{M-n_b(i)}{M}P[\xi_3(n_b(i))] & \text{if the } i^{\text{th}} \text{ new job is of the "write" type, or } n_w(i)=1 \end{cases}$$

(3-A16)

Let $\boldsymbol{n}_w$ be the vector $[n_w(1),n_w(2),\ldots,n_w(\text{q-q'-1})]$. $\boldsymbol{n}_w$ thus indicates the sequence of input job types for those *q-q'*-1 new jobs. Since *w'-w* jobs out of those *q-q'*-1 new jobs are of the "write" type, there are $\binom{q-q'-1}{w'-w}$ possible sequences of input job types. Obviously $\boldsymbol{n}_w \times \boldsymbol{I} = \sum_{i=1}^{q-q'-1} n_w(i) =$ *w'-w* in the case. Therefore, the probability of this type of state change is:

$$a_{11} = a_7 \alpha^{w'-w} (1-\alpha)^{r'-r} \sum_{\substack{\forall \boldsymbol{n}_w \text{ satisfying} \\ \boldsymbol{n}_w \times \boldsymbol{I} = w'-w}} \prod_{i=1}^{q-q'-1} p(i) \tag{3-A17}$$

Case 8: *q>1, q'<q-1, w'≥w+1, r'≥r, (w'-w)+(r'-(r-1))=q-q'*
This case is the same as the previous case with the exception that the HOL job is of the "write" type. Excluding the HOL job, *q-q'*-1 new jobs, or *w'-w*-1 "write" jobs plus *r'*-(*r*-1) "read" jobs are served. The probability of this type of state change is:

$$a_{12} = a_8 \alpha^{w'-w-1} (1-\alpha)^{r'-r+1} \sum_{\substack{\forall \boldsymbol{n}_w \text{ satisfying} \\ \boldsymbol{n}_w \times \boldsymbol{I} = w'-w-1}} \prod_{i=1}^{q-q'-1} p(i) \tag{3-A18}$$

C. Under condition E_3 ("write" job departure)

There are eight cases to consider.

Case 1: *q*≥1, *q'=q, w'=w-1, r'=r*, HOL job is of the "read" type
In this case the HOL job is blocked when a "write" job departs. This occurs when none of the freed disks is the data disk or with probability

$$a_{13} = \frac{\text{number of ways to choose 2 disks out of } 2w+r-1 \text{ busy disks}}{\text{number of ways to choose 2 disks out of } 2w+r \text{ busy disks}} = \frac{\binom{2w+r-1}{2}}{\binom{2w+r}{2}} \tag{3-A19}$$

Case 2: *q*≥1, *q'=q, w'=w-1, r'=r*, HOL job is of the "write" type
Consider three subcases for this case:

Sub-case 2.1: $2w+r<R$
In this sub-case, the number of busy disks before state change is less than *R*. This implies that P[ξ_4]=1 and P[ξ_5] =0. By using the same argument in deriving a_{13}, the probability of this type of state change is the same as a_{13}.

Sub-case 2.2: $2w+r=R$ or $2w+r=R+1$
In this sub-case, both ξ_4 and ξ_5 are possible. P[$\xi_6 | \xi_4$] is given by:

$$P[\xi_6|\xi_4] = \begin{cases} 0 & \text{for } 2w+r\text{-}1<2 \\ \dfrac{\binom{2w+r-1}{2}}{\binom{2w+r}{2}} & \text{for } 2w+r\text{-}1 \geq 2 \end{cases} \tag{3-A20}$$

On the other hand, P[ξ_7| ξ_4]=0 because the number of busy disks is less than R after the "write" job departs. If ξ_5 occurred in the previous state change, it means that all R busy disks were parity disks at that time. Therefore, both ξ_6 and ξ_7 are not possible to occur at the state change. The probability of this type of state change is thus:

$$a_{14} = P[\xi_6|\xi_4]P[\xi_4] \qquad (3\text{-A21})$$

Sub-case 2.3 $2w+r\geq R+2$

This sub-case is the same as sub-case 2.2 with the exception that both P[ξ_7| ξ_4] and P[ξ_7| ξ_5] are not zeros. Given that ξ_4 occurred in the previous state change, ξ_7 happens if i) the data disk is freed at the state change; and ii) all parity disks are busy. P[ξ_7| ξ_4] is then given by:

$$P[\xi_7|\xi_4] = (1 - P[\xi_6|\xi_4])\frac{\binom{2w+r-2}{R}}{\binom{M-1}{R}} \qquad (3\text{-A22})$$

If ξ_5 occurred in the previous state change, it means that all parity disks were busy at that time. For ξ_7 to occur it is necessary that no parity disk is freed at the state change. Therefore, P[ξ_7| ξ_5] in this sub-case is given by:

$$P[\xi_7|\xi_5] = \frac{\binom{2w+r-R}{2}}{\binom{2w+r}{2}} \qquad (3\text{-A23})$$

The probability of this type of state change is:

$$a_{14} = P[\xi_6|\xi_4]P[\xi_4] + P[\xi_7|\xi_4]P[\xi_4] + P[\xi_7|\xi_5]P[\xi_5] \qquad (3\text{-A24})$$

Case 3: $q\geq 1$, $q'=q-1$, $w'=w-1$, $r'=r+1$

In this case, the HOL job blocked in the pervious state change (which is of the "read" type) is served after the state change. This implies that the data disk is freed at the state change. The probability of this type of state change is $a_{15}=1-a_{13}$.

Case 4: $q\geq 1$, $q'=q-1$, $w'=w$, $r'=r$

This is the same as case 3 with the exception that the HOL job is of the "write" type. Therefore, the probability of this type of state change is $a_{16}=1-a_{14}$.

Case 5: $q>1$, $q'<q-1$, $w'=w-1$, $r'=r+(q-q')$

In this case, the HOL job is of the "read" type and q-q'-1 or r'-r read jobs (excluding the HOL job) are served . The probability that q-q'-1 successive jobs are all of the "read" type is $(1-\alpha)^{q-q'-1}$. Similar to the derivations on k_r, the probability of this type of state change is

$$a_{17} = P\begin{bmatrix}\text{The HOL}\\ \text{job is not}\\ \text{blocked}\end{bmatrix} P\begin{bmatrix}q\text{-}q'\text{-1 successive}\\ \text{jobs are all of the}\\ \text{"read" type}\end{bmatrix} P\begin{bmatrix}\text{The first}\\ \text{"read" job is}\\ \text{not blocked}\end{bmatrix} P\begin{bmatrix}\text{The second}\\ \text{"read" job is}\\ \text{not blocked}\end{bmatrix} \ldots P\begin{bmatrix}\text{The } q\text{-}q'\text{-1}^{\text{th}}\\ \text{"read" job is}\\ \text{not blocked}\end{bmatrix}$$

$$= a_{15}(1-\alpha)^{q-q'-1}\left(\frac{M-(2w+r)+1}{M}\right)\left(\frac{M-(2w+r)+1-1}{M}\right)\ldots\left(\frac{M-(2w+r)+1-\{(q-q'-1)-1\}}{M}\right) \qquad (3\text{-A25})$$

Case 6: *q>1, q'<q-1, w'=w-1+(q-q'), r'=r*

In this case, the HOL job is of the "write" type and *q-q'-1* or *w'-w-1* "write" jobs are served. By using the same argument as in the previous case and in the derivations on k_w, the probability of this type of state change is:

$$a_{18} = P\begin{bmatrix}\text{The HOL}\\ \text{job is not}\\ \text{blocked}\end{bmatrix} P\begin{bmatrix}q\text{-}q'\text{-1 successive}\\ \text{jobs are all of the}\\ \text{"write" type}\end{bmatrix} P\begin{bmatrix}\text{The first}\\ \text{"write" job is}\\ \text{not blocked}\end{bmatrix} P\begin{bmatrix}\text{The second}\\ \text{"write" job is}\\ \text{not blocked}\end{bmatrix} \ldots P\begin{bmatrix}\text{The } q\text{-}q'\text{-1}^{\text{th}}\\ \text{"write" job is}\\ \text{not blocked}\end{bmatrix}$$

$$= a_{16}\alpha^{q-q'-1}\left(\frac{M-(2w+r)-2}{M}P[\xi_3(2w+r)]\right)\left(\frac{M-(2w+r)-4}{M}P[\xi_3((2w+r)+2)]\right)$$
$$\ldots\left(\frac{M-(2w+r)-2\{(q-q'-1)\}}{M}P[\xi_3((2w+r)+2\{(q-q'-1)-1\})]\right) \qquad (3\text{-A26})$$

Case 7: *q>1, q'<q-1, w'≥w, r'≥r+1, (w'-w+1)+(r'-r)=q-q'*

In this case, the HOL job is of the "read" type and *q-q'*-1 new jobs (excluding the HOL jobs) which not all of them are of the same type are served. Let $n_w(i)$ be the number of "write" job being brought to the disk array by the i^{th} new job (excluding the HOL job), where i=1,2,3,....*q-q'*-1. Obviously, $n_w(i)$=1 if the i^{th} new job is of the "write" type and $n_w(i)$=0 otherwise. Since in this case *w'-w*+1 "write" jobs are served, $\sum_{i=1}^{q-q'-1} n_w(i)$ =*w'-w*+1. Let $n_b(i)$ be the number of busy disks in the disk array just *before* the i^{th} new job enters the disk array. Thus, $n_b(1)$=2*w*+*r*+1 and

$$n_b(i+1) = n_b(i) + (n_w(i)+1) \qquad (3\text{-A27})$$

The probability that the i^{th} new job is not blocked, denoted as $p(i)$, is given by:

$$p(i) = \begin{cases}\dfrac{M-n_b(i)}{M} & \text{if the } i^{\text{th}} \text{ new job is of the "read" type, or } n_w(i)=0\\[2ex] \dfrac{M-n_b(i)}{M}P[\xi_3(n_b(i))] & \text{if the } i^{\text{th}} \text{ new job is of the "write" type, or } n_w(i)=1\end{cases} \qquad (3\text{-A28})$$

Let $\boldsymbol{n_w}$ be the vector [$n_w(1)$,$n_w(2)$,... ,n_w(q-q'-1)]. $\boldsymbol{n_w}$ thus indicates the sequence of input job types for those *q-q'*-1 new jobs. Since *w'-w* jobs out of those *q-q'*-1 new jobs are of the "write" type, there are $\begin{pmatrix}q-q'-1\\ w'-w+1\end{pmatrix}$ possible sequences of input job types. Obviously $\boldsymbol{n_w}$×$\boldsymbol{I}$=$\sum_{i=1}^{q-q'-1} n_w(i)$ = *w'-w*+1 in the case. Therefore, the probability of this type of state change is:

$$a_{19} = a_{15}\alpha^{w'-w+1}(1-\alpha)^{r'-r-1} \sum_{\substack{\forall n_w \text{ satisfying} \\ n_w \times I = w'-w+1}} \prod_{i=1}^{q-q'-1} p(i) \tag{3-A29}$$

Case 8: ***q>1, q'<q-1, w'≥w, r'≥r+1, (w'-w+1)+(r'-r)=q-q'***

This case is the same as the previous case with the exception that the HOL job is of the "write" type. Excluding the HOL job, *q-q'*-1 new jobs, or *w'-w* "write" jobs plus *r'-r* "read" jobs are served. The probability of this type of state change is:

$$a_{20} = a_{16}\alpha^{w'-w}(1-\alpha)^{r'-r} \sum_{\substack{\forall n_w \text{ satisfying} \\ n_w \times I = w'-w}} \prod_{i=1}^{q-q'-1} p(i) \tag{3-A30}$$

Chapter Four

Dynamic Parity Logging Disk Arrays

RAID (Redundant Arrays of Inexpensive Disks) has gained much attention in the recent development of fast I/O systems. Of the five levels of RAID, the traditional mirrored disk array still provides the highest I/O rate for small "write" transfers. This is because mirrored disk array has no small "write" problem which is found in other levels of RAID. In this paper, we propose a novel RAID architecture for fast engineering database systems, called **Dynamic Parity Logging (DPL) Disk Array**. DPL Disk Array has no small "write" problem and can provide much higher "write" throughput than other RAID architectures when used in engineering database systems. DPL Disk Array also has journalling capability which is very desirable for engineering database systems. In these systems, old versions of designs are usually not removed even though new versions of designs have been completed.

4.1 Introduction

As processor speed continues to increase, the I/O performance of a computer system was recognized to be more and more crucial to the overall system performance [1]. Take for an example, IBM mainframe CPU performance has increase more than 30-fold in the past two decades, whereas IBM disk performance has only doubled in the same period. The I/O performance is even more important to database applications as they are very I/O intensive. Many previous research works on improving the I/O performance for database systems can be found in [2]. Owing to the decreasing memory costs, "memory-resident databases" have been proposed and discussed recently [3-5]. By storing the whole database in fast main memories, the I/O bottleneck is eliminated completely. Datacycle^{TM} architecture [6-7] is a novel technique for fast database machines which has this "memory resident" property. It is expected that distributed database systems on gigabit networks such as the Datacycle^{TM} system will be "memory resident" [8]. However, problems such as failure recovery make "memory resident databases" not practical in many database applications.

Another approach to improve the I/O performance for database systems is to use disk arrays. Redundant Arrays of Inexpensive Disk (RAID) systems were proposed in the late 80's as an alternative to the widely used Single Large Expensive Disk (SLED) systems [9]. Five levels of RAID have been defined when RAID was first introduced. RAID level 5, one of the best performing levels, employs rotated parity with data striped on a unit called a *block* which consists of one or more disk sectors. It can yield very high throughput for large data transfers. For database systems where data transfers are usually small, RAID level 5 also allows data distributed in different disks to be accessed in parallel. However, the throughput reduces significantly if the proportion of "write" transfers increases for such systems. This is because "write" transfers require the extra steps of reading back the old data and the old parity, and the writing in of the new parity. This is commonly called the *small "write" problem* [10]. Moreover, each "write" transfer requires the simultaneous access of two or more disks and thus has a much higher probability of blocking by busy disks than a "read" transfer. The average waiting time for all the required disks to be free in a data transfer, called the *blocking time*, is therefore longer for "write" transfers. Due to these two problems, the traditional mirrored disk array still provides the faster response than RAID level 5 for database applications.

Parity striping [11] is a technique for improving the "write" performance of RAID level 5. It stripes the parity across the disks without striping the data. If the data length of a "write" transfer is greater than the size of a block, parity striping reduces the number of disks involved to only two disks, thereby reducing the blocking time and the expected seek time. However, parity striping provides no performance gain for "write" transfers having data lengths all confined to one block. Since small "write" transfers are usually found in database systems, parity striping is not effective for such use.

Parity logging [12] is another technique which can reduce the "write" transfer overhead by applying journalling techniques. Instead of immediately update the new parity at the end of each "write" transfer, parity logging buffers the parity update image in a fault tolerant buffer. When enough parity update images are buffered to allow for an efficient disk transfer, they are written to a log disk. When the log disk is full, parity reconstruction of the whole disk array is performed.

LRAID-X4 [13] is a scheme similar to parity logging. It uses separate parity and parity update log disks, and periodically applies the logged updates to the parity disk. Another technique worth noting is the *floating data and parity* modification to RAID level 5 [14]. It restricts individual cylinders of a disk to contain either data or parity, but not both. Part of the storage of the cylinder is reserved as free space. When there is a "write" transfer, the new data or the new parity can be written to the free space of the targeted cylinder immediately after the old contents have been read. This technique effectively reduces the extra rotational delay for "read-modify-write" accesses. However, it does not reduce the blocking time for data transfers.

The above survey shows extensive research efforts on improving the "write" performance of RAID in conventional database systems. However, similar research on specific types of database systems are relatively few. In this chapter, we focus on EDS and discuss the design of a fast disk array architecture for these systems. As discussed in [15-20], EDS differs from conventional database systems in many aspects. It is beyond the scope of this chapter to discuss all of their differences. Instead, we discuss in the following the unique ways of processing data and the unique data storage requirements for EDS. First, I/O requests in EDS are usually processed in serial and there is no need for special concurrency control mechanism in the storage subsystems [pp.69, 16]. Due to the serial processing of I/O requests, blocking due to busy disks becomes a decisive factor on system performance in EDS. This is because a request blocking will not only delay the I/O operations of the blocked request, it will also delay the I/O operations of all subsequent requests where some may target on other disks which are free. Second, the requirement on data availability for EDS need not be as high as conventional database systems. Once a design is loaded to the workstation from the file server, the file server can tolerate temporarily suspensions of I/O services in case of a disk failure. Third, older versions of an engineering design are rarely discarded [pp.261-262, 17]. Therefore, the support of journalling in data storage subsystem is very desirable. The support of journalling also facilitates the implementation of some common design functions such as *undo* and *redo*. Finally, the I/O rate required by EDS is substantially lower than on-line transaction processing systems where some may require an I/O rate of over 1000 transactions per second.

On the other hand EDS is similar to conventional database systems in the following aspects. First, high data reliability is required by EDS. EDS should provide archiving facilities for archiving data to tertiary storage [pp.107, 18]. Regular system backup should also be performed to protect the lost of valuable data. Second, fast response time for "write" requests is required for EDS. Since transactions in engineering databases are usually very long, data updates at different *savepoints* of a transaction are performed continuously to protect the loss of data during the transaction [pp. 71-72, 16 & pp.105-106, 18]. Third, the data volume of EDS is usually very large [pp.262, 17] and so disk arrays is well suited for its use.

In this chapter a new RAID architecture called **Dynamic Parity Logging (DPL) Disk Arrays** is proposed for fast EDS. DPL Disk Array aims at solving the small "write" problem and reducing the blocking time for "write" transfers. It can provide much faster "write" response than mirrored disk array, while maintaining the same high storage utilization of RAID level 5. Although both DPL Disk Array and parity logging apply journalling techniques, the expected length for the former is much shorter. In the next section, we describe DPL Disk Array in detail. This is followed by a section on the performance study of DPL Disk Array. The superior performance of DPL Disk Array is then concluded from the results.

4.2 DPL Disk Array Architecture

A. Block Coordinate System

The block coordinate system can be used to describe RAID operations, including the DPL Disk Array. Consider a disk array system with a total of N+1 disks. Let each disk has K blocks and each block may consist one or more sectors. We denote block j in disk i as $B(i,j)$, and its contents as $b(i,j)$. A block can store either data or parity. We define a *parity set* to be a collection of blocks for which the parity sum of these blocks is always maintained to be "1" for odd parity and "0" for even parity. Let there be a total of X parity sets in the disk array denoted as A_1,A_2,...,A_X,. Take mirrored disk array or RAID level 1 as an example, a pair of mirrored disks has K parity sets and each parity set has exactly two blocks. Let disks i and i+1 be a pair of mirrored disks where i is odd. Then the parity sets in this disk array can be described as:

$$A_j = \{B(i,j), B(i+1,j)\} \qquad i = 1,3,\ldots,N-1,\ j = 1,2,\ldots,K \tag{4-1}$$

For each parity set A_j, we have

$$b(i,j) = b(i+1,j) \qquad i = 1,3,\ldots,N-1,\ j = 1,2,\ldots,K \tag{4-2}$$

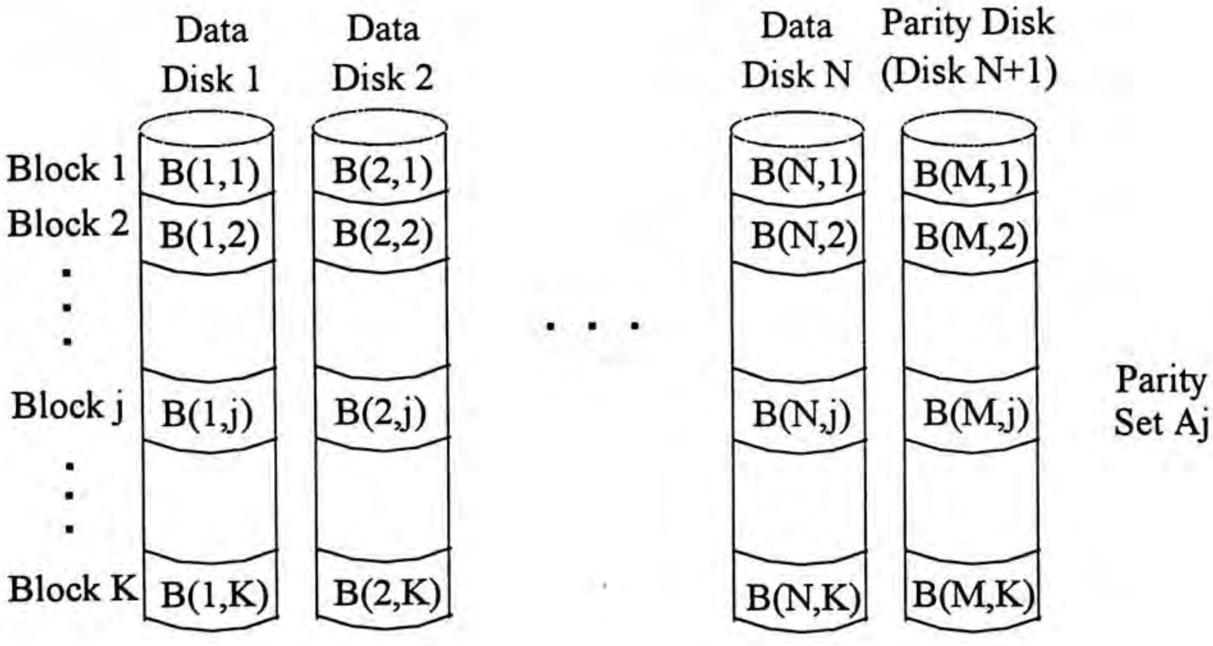


Figure 4-1. The block coordinate system showing RAID level 4.

Similarly, RAID level 4 or level 5 has K parity sets which can be described as:

$$A_j = \{B(1,j), B(2,j), \ldots, B(N,j), B(N+1,j)\} \qquad j = 1,2,\ldots,K$$

and (4-3)

$$\sum_{i=1}^{N+1} b(i,j) = \begin{cases} (1,1,1,\ldots,1) & \{\text{odd parity}\} \\ (0,0,0,\ldots,0) & \{\text{even parity}\} \end{cases}$$

where Σ is defined here as the mod-2 sum of the block contents. Figure 4-1 shows RAID level 4 by using the block coordinate system. As shown, parity set A_j is formed by the data blocks having the same block number *j* plus a *parity block* holding the parity sum of the data blocks. The parity blocks are always saved in disk *N*+1 for RAID level 4.

B. Overview of DPL Disk Array

Observe that mirrored disk array systems do not have the small "write" problem. "Write" transfers are performed by directly replacing the old contents with the new ones. The average service time for a "write" transfer is just the average total time of two independent "write" operations performing on two disks, and is slightly longer than the average time of a "read" transfer. Another observation is that regular system backup is performed on most EDS. Data which has not been modified since the last system backup can be treated as reliable as it can be restored from the tertiary storage when disk failures occur. Therefore if the *updated* data are protected from disk failures by redundancy adding, data reliability for all data in the disk array is ensured. Although the need of data restoration from the tertiary storage makes the system not highly available, this is not a serious problem for EDS as mentioned above. We shall later see that the time required to restore a disk is small if fast optical disks are used for tertiary storage. These two observations lead us to the design of DPL Disk Array.

The configuration of DPL Disk Array is shown in Figure 4-2. As shown a DPL Disk Array has a total of *N*+2 disks. Similar to RAID level 4, disks 1 to *N* are for data storage and are called the *data disks*. Disks *N*+1 and *N*+2 are mirrored disks and are called the *parity disks*. This pair of disks is used to store the parity blocks of parity sets. Unlike the static assignment of parity sets found in the other RAID architectures, parity sets in a DPL Disk Array are *dynamically* assigned when block updates are performed.

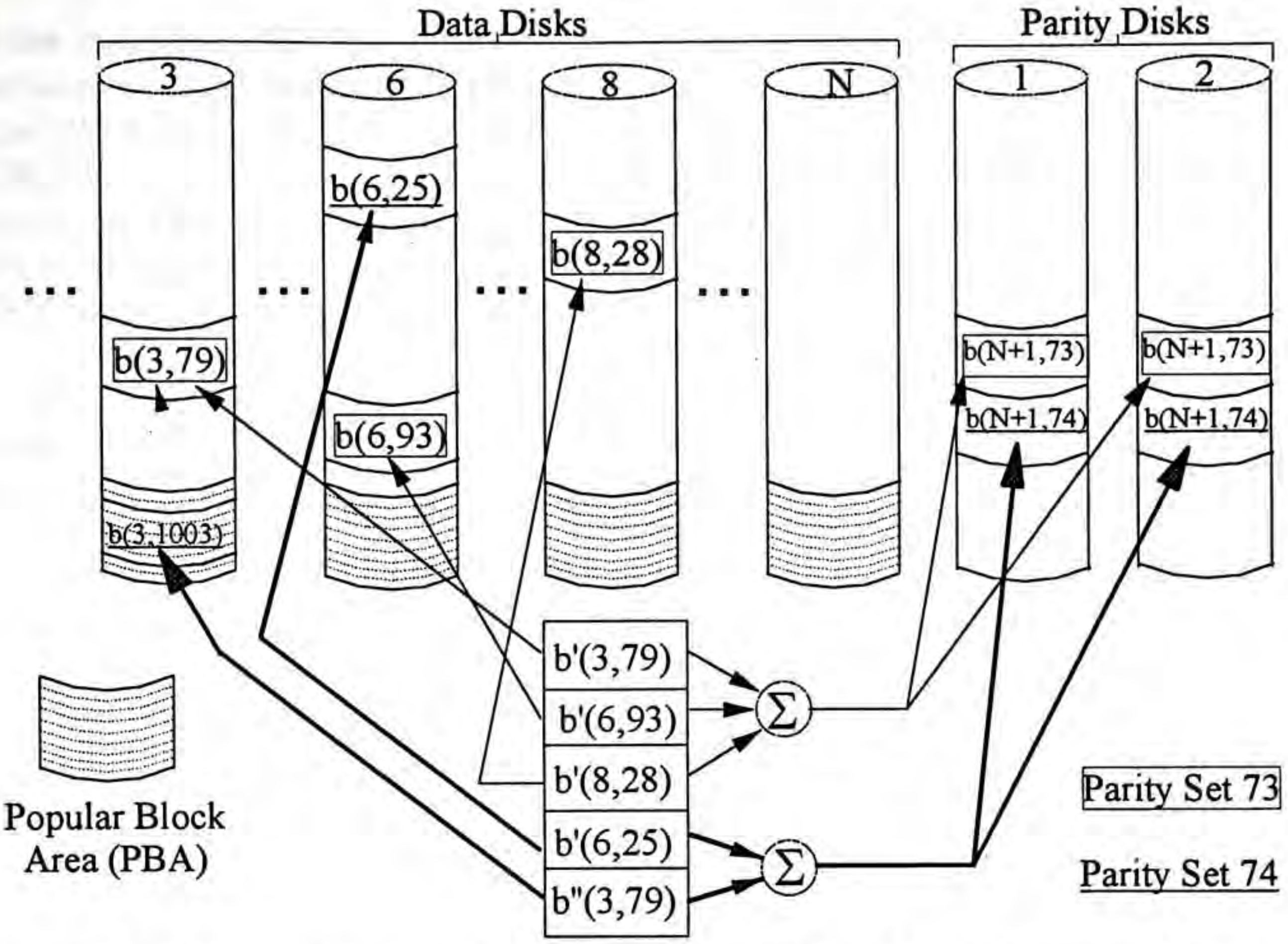


Figure 4-2 An example showing the working principle of DPL Disk Array for "write" transfers.

The working principle of DPL Disk Array for "write" transfers is illustrated in Figure 4-2 which shows *five* successive updates on *four* different blocks $B(3,79)$, $B(6,93)$, $B(8,28)$ and $B(6,25)$. Suppose the contents of these blocks have not been modified since the last system backup. The new contents for the five updates are identified as $b'(3,79)$, $b'(6,93)$, $b'(8,28)$, $b'(6,25)$ and $b''(3,79)$ respectively as shown in the figure. Noting that $B(6,93)$ and $B(6,25)$ are located in the same disk, and $B(3,79)$ is modified twice. Since second copies of the original block contents $b(3,79)$, $b(6,93)$, $b(8,28)$ and $b(6,25)$ have been stored in the tertiary storage, they can be restored if necessary. These blocks can therefore be overridden by new contents. The new contents however should be protected by additional parity. Therefore, the mod-2 sum of the new contents of the first three updates, i.e. $b'(3,79) \oplus b'(6,93) \oplus b'(8,28)$, is computed and saved to the parity disks as a parity block. Figure 4-3 shows the structure of a parity block. It consists of a block header which holds the block identifiers of its member blocks, and a parity sum of its member blocks. The parity block for the first 3 updates of our example is shown in Figure 4-3b. A parity set consisting of three data blocks and 1 parity block is thus formed. This parity set is identified as parity set 73 in Figure 4-2. These blocks can be assigned to the same parity set because they are all located on different data disks. If one of these data disks fails, the lost blocks in the failed disk can be recovered from the remaining blocks of the parity set. This parity set cannot include the next updated block $B(6,25)$ because the parity set already has a member block $B(6,93)$ in disk 6. We can, however, form the next parity set 74 starting from this block update. In our example, parity set 74 consisting of $B(6,93)$, and $B(3,1003)$, and their parity sum $b'(6,25) \oplus b''(3,79)$ is saved to the parity disks (Figure 4-

3c). Note that block $B(3,79)$ was updated before. Its current contents $b'(3,79)$ therefore cannot be overridden. Instead, we store $b''(3,79)$ to a reserved area of the disk called the *Popular Block Area (PBA)*[1] at location $B(3,1003)$. As shown in Figure 4-2 each data disk has a PBA for storing new updates of popular blocks in that disk. When a popular block is written to the PBA, a block header pointing to its original location on the disk is also written (see Figure 4-4). This pointer is used for restoring the content of the block when system backup is performed.

If there is only one parity disk, all parity information is lost when the parity disk fails and a very time consuming backup of all data disks is required. This can be avoided by using a pair of mirrored disks to store the parity information.

Entry 1	Entry 2	. . .	Entry N	Entry N+1
Block identifier of the first member block	Block identifier of the second member block	. . .	Block identifier of the Nth member block	Parity Sum S

a) Parity block format

B(3,79)	B(6,93)	B(8,28)	empty	. . .	S=b(N+1,73)=b'(3,79) ⊕ b'(6,93) ⊕ b'(8,28)

b) Parity block of parity set 73

B(6,25)	B(3,1003)	empty	. . .	S=b(N+1,74)=b'(6,25) ⊕ b"(3,79)

c) Parity block of parity set 74

Figure 4-3 Structure of parity blocks showing the example given in Figure 4-2.

[1] Blocks which are updated more than once after backup are called the *popular blocks*.

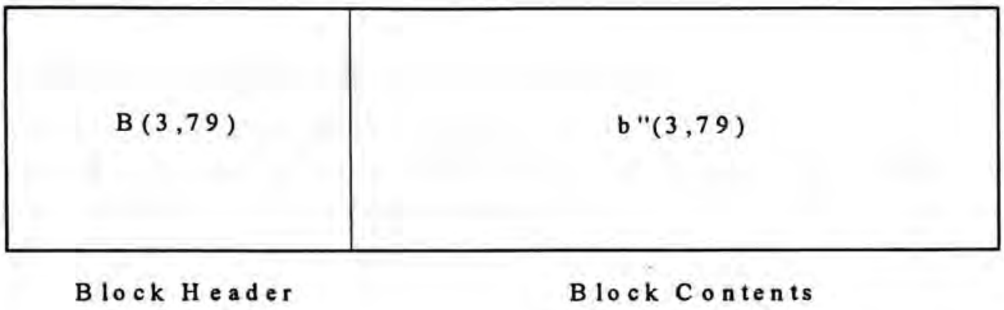


Figure 4-4 The structure of a popular block in the APB showing block B(3,1003).

Some interesting properties of DPL Disk Array are observed from the discussions given above. We shall discuss them first before presenting the detail operation of DPL Disk Array.

C. Properties of DPL Disk Array

1. These is no small "write" problem.

This is because blocks are written to the disks without first reading back the old data or parity.

2. The blocking time for "write" transfers is much smaller than that of other RAID architectures.

This is because a "write" transfer in DPL Disk Array is performed on one data disk only whereas other RAID architectures requires two disks. DPL Disk Array can therefore provide much better "write" performance than other RAID architectures. This will be verified in section 4.4.

3. The parity disks contain a journal of block updates.

As discussed above, the update blocks are assigned to parity sets according to their arrival sequence. Since the sequence of updates within the same parity set is also known from the sequence of block identifiers in the block header of the parity block, the parity disks contain full journal of block updates.

4. Data will not be lost in single disk failures.

This property will be apparent when we introduce the procedures of recovering a failed data disk in section 4.3. When one of the parity disks fails, no data except the parity information is lost.

5. The log volume will be significantly smaller than the updated volume.

Since a parity set contains multiple updated blocks, the number of parity blocks will be much smaller than the number of updated blocks. That is to say, the log volume will be significantly smaller than the update volume. The actual ratio between their sizes depends on the average size of the parity sets and will be derived in section 4.4.

6. The parity disks work sequentially under all conditions.
We mentioned before that new parity blocks are written to the parity disks sequentially. When a parity disk is read for data recovery, it is also read out sequentially to provide the necessary parity information. Therefore, sequential access devices such as optical disks or tape drives can also be used to store the parity blocks.

7. No data is lost when the I/O controller fails.
When the I/O controller fails, no data saved on disks is lost. The contents of the tables and counters used by the I/O controller can be stored in non-volatile memory for fast resume of I/O operations when the I/O controller is up again.

8. DPL Disk Array requires a brief period of data restoration in case of disk failures.
When disk failures occur, the contents of the failed disk must be recovered by first loading the original data from the tertiary storage and data in the failed disk is not available during data restoration. However, the data restoration time may be very short with the use of today's optical storage technology. Take for example, a 12cm CD-ROM drive can deliver data at continuous rate of 1.5MB/s [21]. At this rate, restoring a 1GB disk takes only 11 minutes. The restoration time can further be shorten if larger diameter optical disks are used. Nevertheless this property limits the application of DPL Disk Arrays to EDS-like systems only.

4.3 DPL Disk Array Operation

Figure 4-5 shows the organization of DPL Disk Array. Requests for data transfers are sent from host to the I/O controller for I/O operations. The I/O controller consists of seven parts: an I/O queue, a local memory, an I/O processor, a DMA controller, a block location table, N data disk counters, and a parity disk counter. The I/O queue is used for storing I/O requests and the requests are served in a FCFS manner. The data associated with a request (i.e. the new data for a block) is stored in the local memory when the request is placed on the queue. The local memory is also used for buffering the data sent to or read from each of the disks and for storing all temporary data for processing. The DMA controller functions as a multiplexer/demultiplexer and handles simultaneous data transfers to various disks. The I/O processor is the heart of the I/O controller. It is responsible for distributing works to the disks for data transfers. It also performs necessary data processing works such as parity computation. The block location table is used for locating data blocks in the disk array. As shown in Figure 4-6 it is a $K_D \times N$ table, where K_D is the number of data blocks per disk. Since there are K blocks in a disk, the size of the PBA is therefore equal to $K_P=K-K_D$ blocks. Each data block has its own entry in the table which consists of a status bit and a block address. For block $B(i,j)$ the contents of the status bit and the block address are denoted as $s(B(i,j))$ and $a(B(i,j))$ respectively. The status bit $s(B(i,j))=1$ indicates that the block has been updated since last system backup and $s(B(i,j))=0$ indicates otherwise. The block address points to the current location of the block in the disk array. In particular, $a(B(i,j))=0$ indicates that $b(i,j)$ is the most recently updated contents of $B(i,j)$ and $a(B(i,j))$ points to a location in the PBA indicates that $B(i,j)$ has been updated more than once. One thing worth noting is the size of the block location table. For a typical disk having 2^{19} blocks (which corresponds to a 1 GB drive with a block size of 2KB), each entry in the table require 19+1 bits. The size of the block location table for a N=25 disk array system is $20 \times 25 \times 2^{16}$ bytes, or 32MB. By using today's flash RAM technology, it is practical to implement it on non-volatile memory units [22]. To facilitate the management of PBA, the I/O controller maintains a pointer for each data disk pointing to the next available block in the PBA. We denote the pointer value for data disk i by c_i, where $K_D < c_i \leq K$. Each time the I/O processor attempts to write a popular block to PBA, the corresponding data disk pointer is read to give the appropriate location for the block. That pointer value is then incremented by 1 to point to the next available block in the PBA. A similar pointer c_p is maintained for the parity disks which points to the location for the next parity block writing.

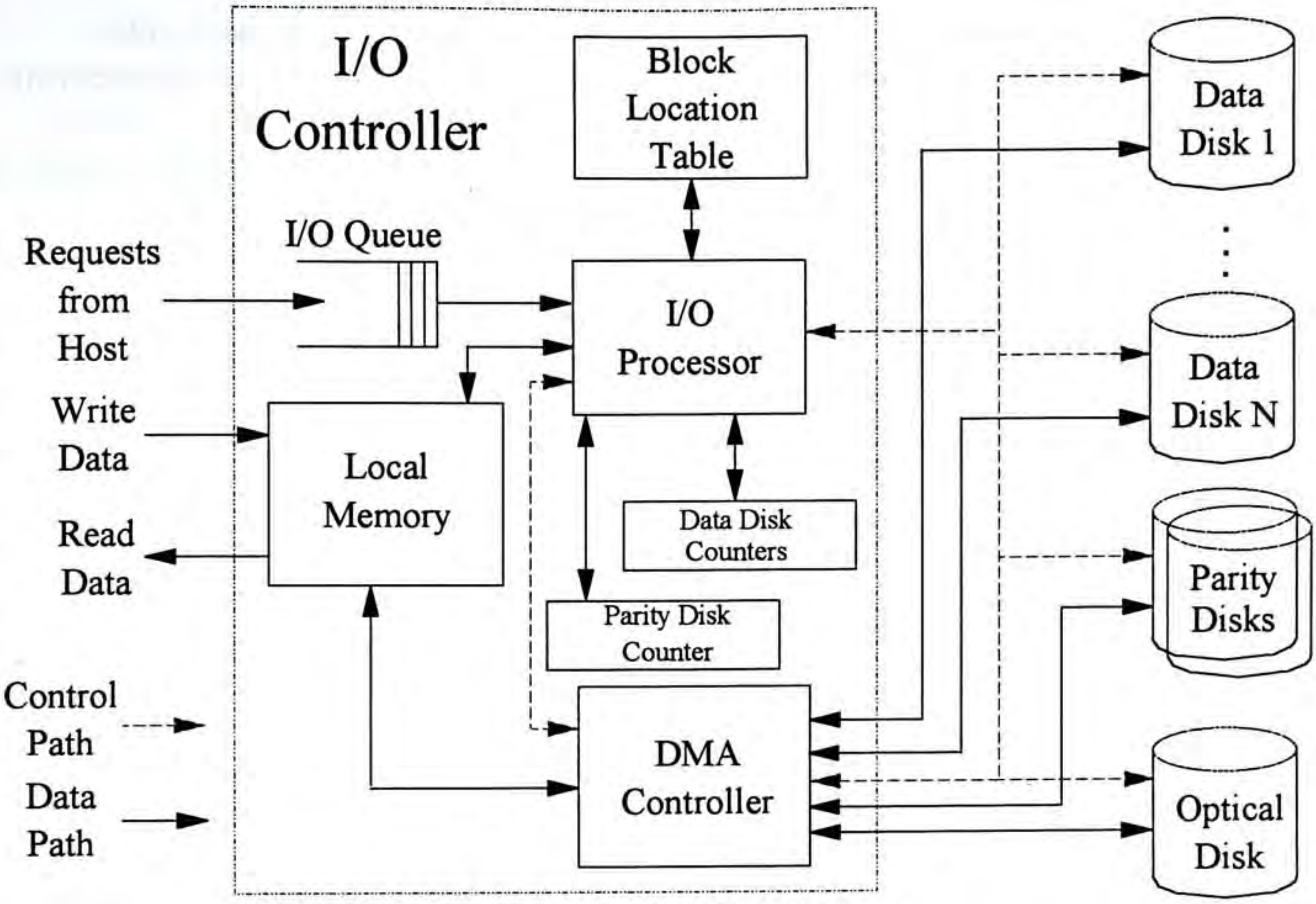


Figure 4-5 Organization of DPL Disk Array.

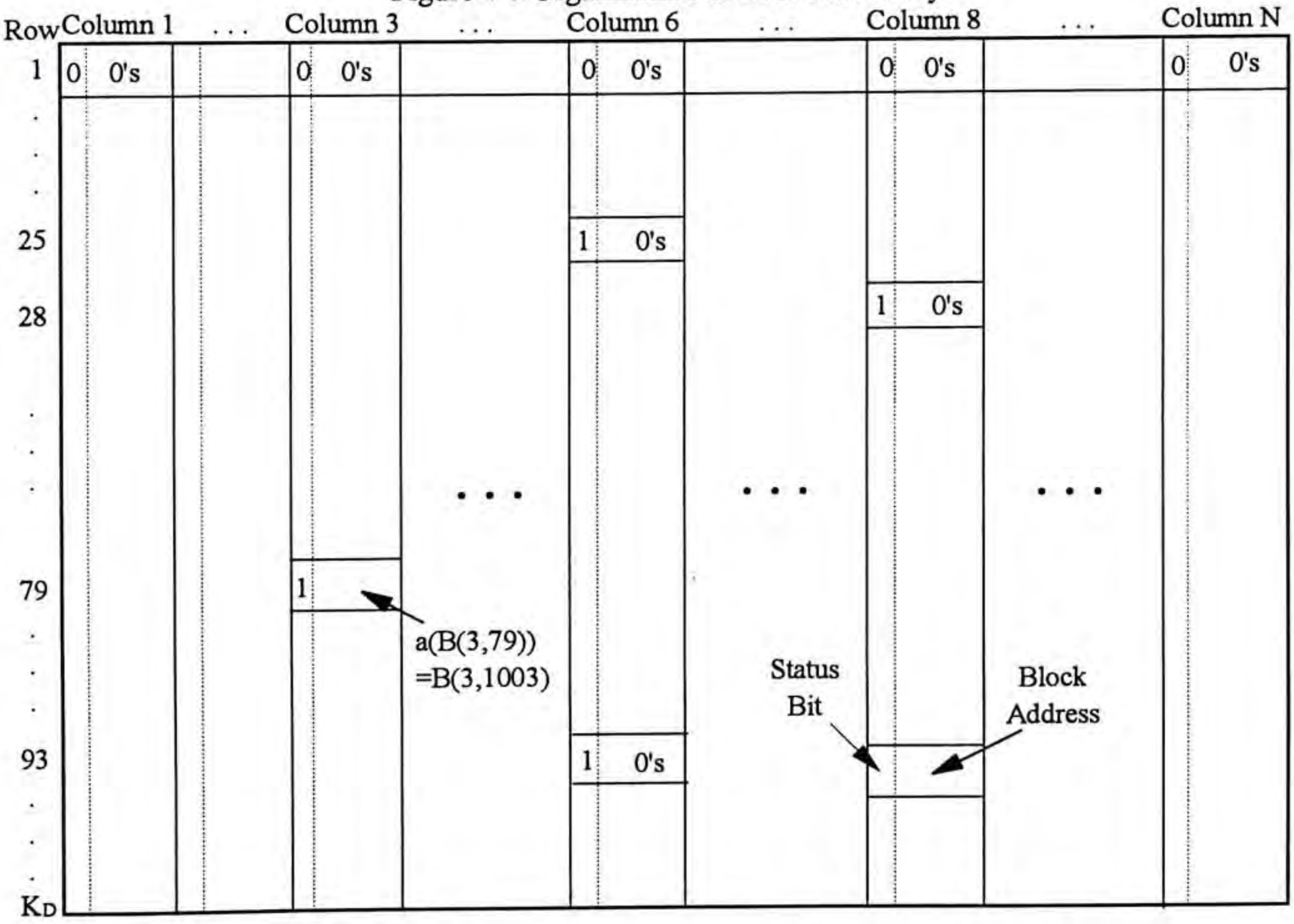


Figure 4-6 Block location table for the example given in Figure 4-2.

DPL Disk Array operates in three modes: system backup mode, normal mode and data recovery mode. These modes of operations are briefly discussed below.

A. System Backup Mode

This mode can be invoked regularly or when either the parity disks or the PBA are full. Under this mode, only the "read" transfers can be performed. It operates as follows:

1. Disk images of all the *N*+1 disks are copied to the tertiary storage one by one. The tertiary storage is shown as an optical disk in Figure 4-5.

2. The contents of the popular blocks in the PBA are copied back to their original locations in the disk array.

3. All pointers and the block location table in the I/O controller are cleared.

The pseudo codes which show the detail operations in this mode is given in appendix A.

B. Normal Mode

DPL Disk Array works in this mode under normal condition.

1. When a "read" request arrives:
   i) The I/O processor places the request to the request queue.

2. When a "write" request arrives:
   i) The I/O processor checks whether the new block associated with the request can be assigned to the parity set currently under construction or not.
   ii) If yes, a new parity sum for the parity set is computed and saved to the non-volatile memory.
   iii) If no, the "write" data is saved to the local memory and a new parity set is created and stored to the non-volatile memory.
   iv) The I/O processor then places the request to the request queue.

3. When a busy disk finishes its operation:
   i) If the request queue is not empty, the I/O processor checks whether the request which is at the top of the queue targets on the disk just freed or not.
   ii) If not, no further processing is performed.
   iii) If the request targets on the disk just freed, the I/O processor removes the request from the queue and performs the following operations according to the type.
   iv) If the request is of the "read" type, the I/O processor
      a) instructs the DMA controller and the data disk involved to load the requested block to the local memory (the address for the requested block is given by the block location table); and
      b) signal to host for "read" completion.
   iv) If the requests is of the "write" type, the I/O processor

a) instructs the DMA controller and the data disk involved to store the new data of the request to the disk (the location of storage is given by the block location table);
b) updates the target block's entry in the block location table;
c) if the request creates a new parity set, saves the parity sum of the preceding parity set (which is stored in the non-volatile memory) to the parity disks; and
d) signals the host for "write" completion.

The pseudo codes which show the detail operations in this mode is given in appendix B. As shown from the above operations, assigning new blocks to parity sets and computing parity sum can be performed when a "write" request arrives.

C. Data Recovery Mode

The system is switched from the normal mode to this mode when disk failures occur. The operations in this mode are outlined below.

1. The failed disk is either fixed or replaced by a spare disk.
2. The original contents of the failed disk at last system backup are restored from the tertiary storage.
3. The updated data after last system backup can be recovered by:
   a) sequentially search one of the parity disks for parity blocks having member blocks in the failed disk.
   b) If found, recover the contents of the updated blocks in the failed disk. This can be done as follows. Suppose parity set $A_j=\{B(d_1,o_1),B(d_2,o_2),\ldots,B(d_i,o_i),\ldots,B(d_k,o_k)\}$ has one member block $B(d_i,o_i)$ in the failed disk i. Assuming that even parity is used, the contents of $B(d_i,o_i)$ can be recovered by:

   $$b(d_i,o_i) = \sum_{x=1,x\neq i}^{k} b(d_x,o_x) \qquad (4\text{-}4)$$

   c) After finish searching the parity disk, check the parity set which was in construction when the disk failed to see whether it contains member block in the failed disk or not. If found, recover the contents of the updated block in the failed disk.

The pseudo codes which show the detail operations in this mode is given in appendix C. As shown from the above operations, data can be recovered if single disk failure occurs. This proves property 4 of DPL Disk Array stated earlier.

4.4 Performance of DPL Disk Array

Figure 4-7 shows the queueing model for a DPL Disk Array. Requests for data transfers are sent from host and become *jobs* for the servers. Job arrivals are assumed to be a Poisson process with rate λ, and are assumed to target uniformly on all disks. The probability that a job targets on a particular block of a disk is also assumed to be the same for all blocks. A job is of the "write" type with probability α and of the "read" type with the remaining probability. As we have mentioned before, one parity block is written to the parity disks only when the system finishes the construction of a parity set. Whe a parity set is still in construction, the parity sum of the parity set is kept in the non-volatile memory. Therefore, we only need to consider the data disk operation for each "write" job because the access time of the non-volatile memory is much faster than that of the data disks. For mathematical convenience we assume that the service time for a job is exponentially distributed with service rates μ_w and μ_r for "write" jobs and "read" jobs respectively. We also assume that the I/O controller is fast enough so that it does not affect the system performance.

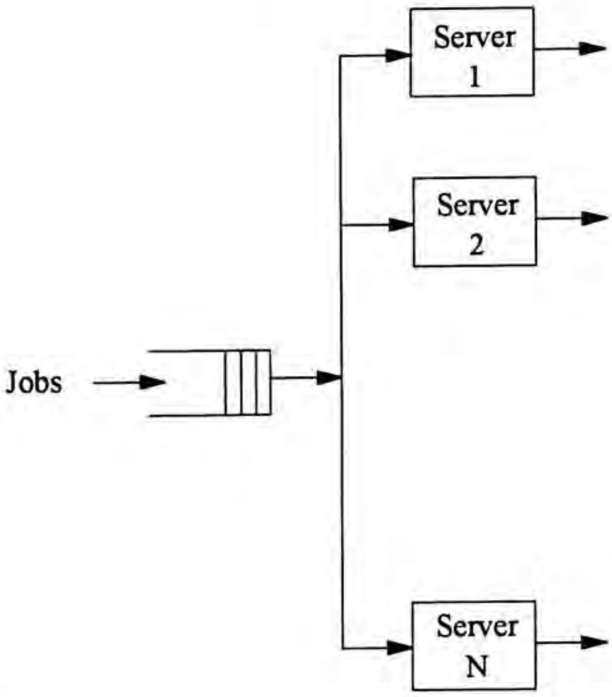


Figure 4-7 Queueing models for DPL Disk Array.

A) Mean System Backup Time

Suppose the system finishes backing up the data and switches to normal mode at time 0. All blocks are marked as unchanged at that time. Since all blocks are requested with the same probability, the rate of "write" arrivals targeting on a particular block γ is given by:

$$\gamma = \frac{\text{Rate of " write" arrivals to the disk array}}{\text{Total number of blocks in the disk array}} = \frac{\alpha\lambda}{NK_D} \tag{4-5}$$

Let p_k be the probability that there are k "write" requests in $(0,t)$ targeting on a particular block. With the Poisson arrival assumption, we have

$$p_k = \frac{(\gamma t)^k}{k!} e^{-\gamma t} \tag{4-6}$$

Consider block i in disk j. Let $X_i(t)$ be the number of block i images written to PBA of disk j in $(0,t)$. For k+1 arrivals k images are written to PBA, we have

$$P[X_i(t) = k] = \begin{cases} p_0 + p_1 & \text{for } 0 \le k \le 1 \\ p_{k+1} & \text{for} k \ge 2 \end{cases} \tag{4-7}$$

The expected value of $X_i(t)$ is given by:

$$E[X_i(t)] = \gamma t - (1 - p_0) \tag{4-8}$$

The variance of $X_i(t)$ is given by:

$$VAR[X_i(t)] = \sum_{k=0}^{\infty} P[X_i = k]k^2 - E[X_i]^2 = [\gamma t - (1 - p_0)]^2 \tag{4-9}$$

Next, let random variable $X(t)$ denotes the total number of blocks written to PBA of disk j in $(0,t)$, or

$$X(t) = \sum_{i=1}^{K_D} X_i(t) \tag{4-10}$$

As the X_i's are independent and identically distributed random variables, by central limit theorem the distribution of $X(t)$ can be well approximated by a Gaussian distribution with mean $m=K_D[\gamma t-(1-p_0)]$ and variance $\sigma^2=K_D[\gamma t-(1-p_0)]^2$ if K_D is not too small. Therefore, the probability $r(t)$ that the PBA of disk j does not overflow in $(0,t)$ is given by:

$$r(t) = P[X(t) \le K_p] = \int_{-\infty}^{K_p} \frac{e^{-(x-m)^2/(2\sigma^2)}}{\sqrt{2\pi\sigma}} dx \tag{4-11}$$

Finally, the probability that the PBA's of all N disks do not overflow in $(0,t)$ is simply given by $r^N(t)$. Note that system backup should be performed if PBA of any one of the disks overflows. If we let random variable T be the time which the system has to perform system backup, the mean system backup time T_B is given by:

$$T_B = \int_0^{\infty} (1 - P[T \le t])dt = \int_0^{\infty} (1 - (1 - r^N(t)))dt = \int_0^{\infty} r^N(t)dt \tag{4-12}$$

As an example, consider a DPL Disk Array with N=10 and K_D=50,000. Figure 4-8 plots the mean system backup time T_B (in days) against the "write" arrival rate $\alpha\lambda$ for three different values of K_P. Simulation was also performed to verify the analysis given above. We observe from the figure that both the analytic and simulation results match very well with each other. Note that for all simulation results shown in this chapter we have extended the simulation time sufficiently long to make the 95% confidence intervals smaller than the size of the markers shown. Another observation is that T_B is inversely proportional to $\alpha\lambda$ for all values of K_P shown. We also observe that T_B is increased by about 45% when the size of the PBA is doubled for all "write" arrival rates shown.

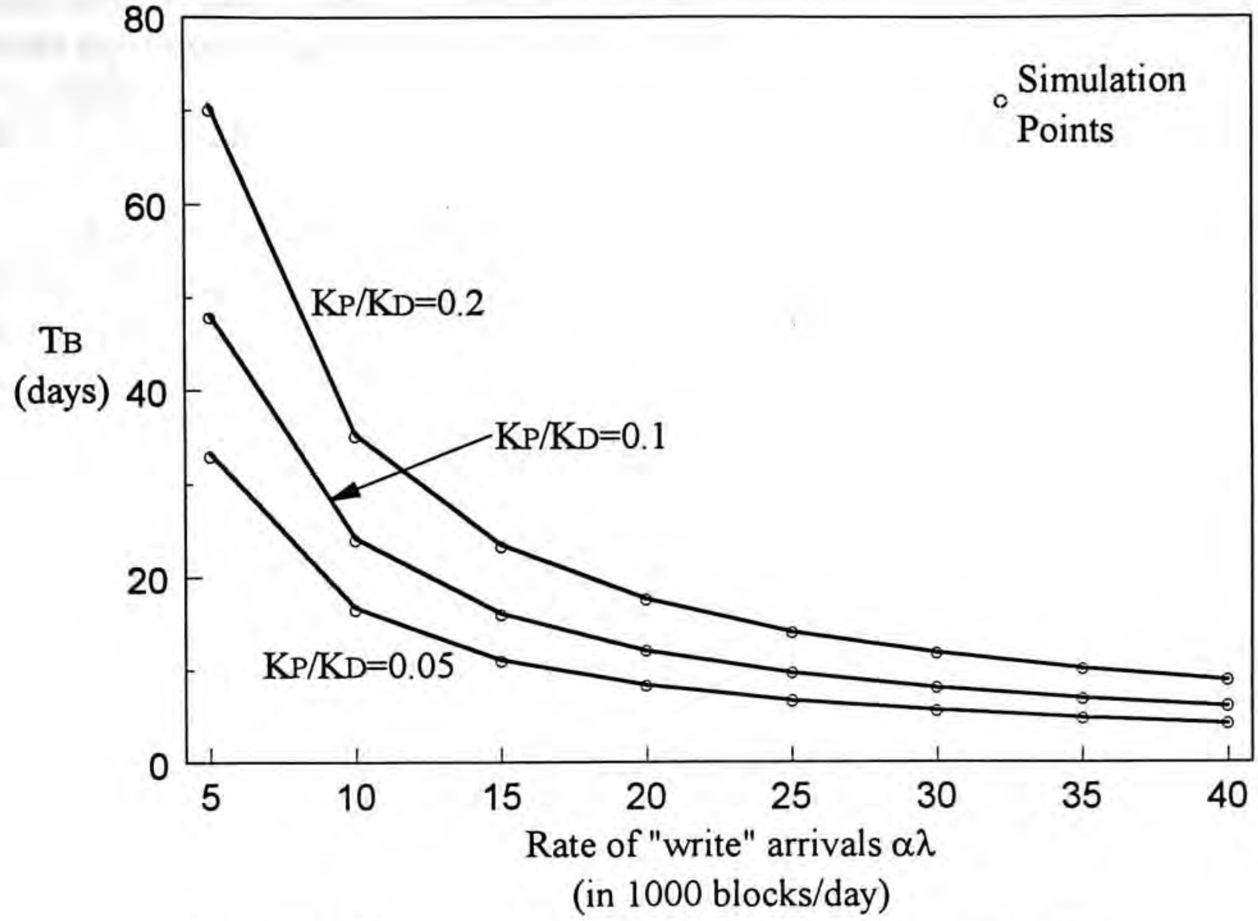


Figure 4-8 Mean system backup time against the rate of "write" arrivals.

If T_B and the maximum number of blocks being written between successive system backups are given, K_P can be determined from Figure 4-8. Take for an example, if T_B=7 (i.e. weekly backup is performed) and the maximum number of block updates per day is 25,000 blocks, the minimum value of K_P should be $0.05K_D$ or 2,500 blocks. If the maximum number of block updates per day grows to 40,000 blocks, K_P should be at least $0.1K_D$ or 5,000 blocks.

B) Utilization of the parity disks

Consider a particular parity set A_s and let $|A_s|$ be the number of elements in A_s. Then

$$P[|A_s|=m]=P\begin{bmatrix}\text{The first}\\\text{block is}\\\text{new to }A_s\end{bmatrix}P\begin{bmatrix}\text{The second}\\\text{block is}\\\text{new to }A_s\end{bmatrix}\ldots P\begin{bmatrix}\text{The }m^{\text{th}}\\\text{block is}\\\text{new to }A_s\end{bmatrix}P\begin{bmatrix}\text{The }(m+1)^{\text{th}}\\\text{block is not}\\\text{new to }A_s\end{bmatrix} \tag{4-13}$$

$$=1\times\frac{N-1}{N}\times\frac{N-2}{N}\times\ldots\times\frac{N-(m-1)}{N}\times\frac{m}{N}=\frac{m(N-1)!}{N^m(N-m)!}$$

The expected size L_s of parity set A_s is therefore

$$L_s=\sum_{m=1}^{N}mP[|A_s|=m]=\sum_{m=1}^{N}\frac{m^2(N-1)!}{N^m(N-m)!} \tag{4-14}$$

Since only one parity block is generated for each parity set, the average number of parity blocks generated in T_B, denoted as B_p, is therefore

$$B_p = \frac{\alpha \lambda T_B}{L_s} \tag{4-15}$$

Figure 4-9 shows L_s against N. For a small disk array of 4 data disks, we find that the log volume is about 45% of the update volume ($1/L_s \approx 0.45$). For a large disk array of 24 data disks, the log volume reduces to about 17% of the update volume ($1/L_s \approx 0.17$).

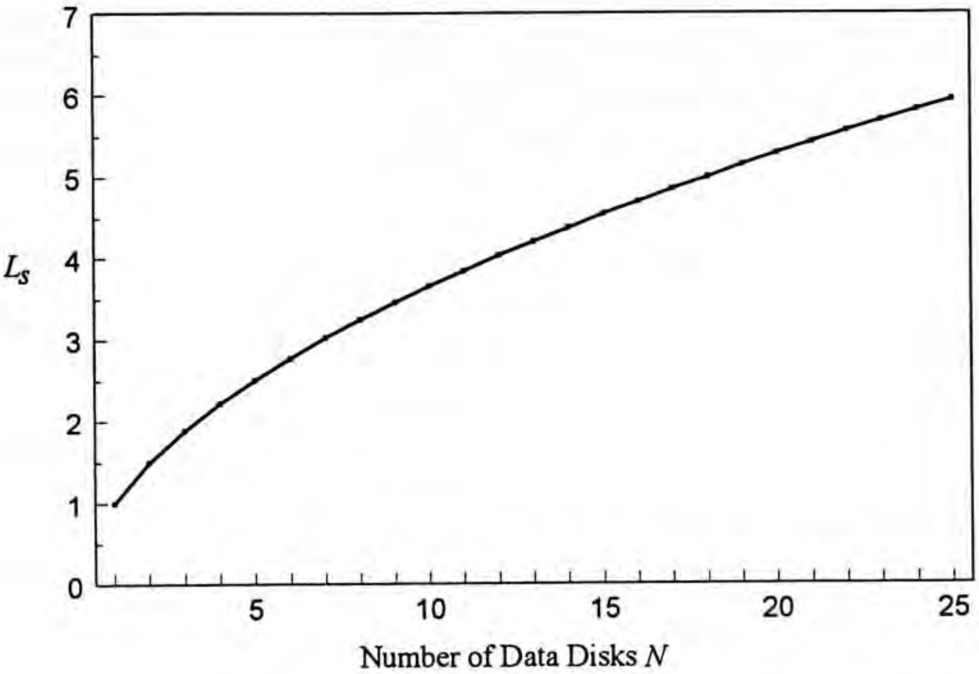


Figure 4-9 The average number of blocks in a parity set (excluding the parity block) grows with the number of data disks.

C) Average Delay

As discussed before, DPL Disk Array has no small "write" problem. This is because blocks can be directly written to the disks without reading back the old parity and old data. Therefore, it is reasonable to assume that the same average delay is experienced by the "write" jobs and "read" jobs. To obtain the average job delay for DPL Disk Array, we first consider the operations of RAID level 5 when all jobs are of the "read" type (i.e. α=0). Since all jobs are of the "read" type, each of them requires the access of only one data disk. Note that for DPL Disk Array each job also requires the access of only one data disk. If the operations of each disk is modelled by an exponential server, the queueing model for RAID level 5 when α=0 is almost identical to that of DPL Disk Array (see Figure 4-7) The two models differ only in the number of servers (N+1 for RAID level 5 and N for DPL Disk Array) and the service rates. A detail analysis of RAID level 5 using exponential servers is reported in chapter 3 of this thesis. By using the same analysis we obtain the average delay for DPL Disk Array.

Figure 4-10 shows the average job delays for DPL Disk Array and RAID level 5 when each architecture has four data disks. We assume in this example that $\mu_w = \mu_r = 50$ jobs/sec for DPL Disk Array. For RAID level 5, the service rates for "write" jobs and "read" jobs are assumed to be 30 jobs/sec and 50 jobs/sec respectively. Since "write" operations in RAID level 5 need to read back old parity and data, the corresponding service rate is lower due to extra disk rotation. We find from Figure 4-10 that DPL Disk Array outperforms RAID level 5 for most values of α. This is because there is no small "write" problem in DPL Disk Array and the "write" jobs for DPL Disk Array only require the access of one disk. When α=1, DPL Disk Array provides maximum throughput which is 2.5 times higher than that of RAID level 5. When all the jobs are of the "read" type, we observe that RAID level 5 performs slightly better than DPL Disk Array because data is distributed into 5 disks for RAID level 5 and is only distributed into 4 disks for DPL Disk Array. The difference in performance for α=0 will be smaller for larger N's for obvious reason.

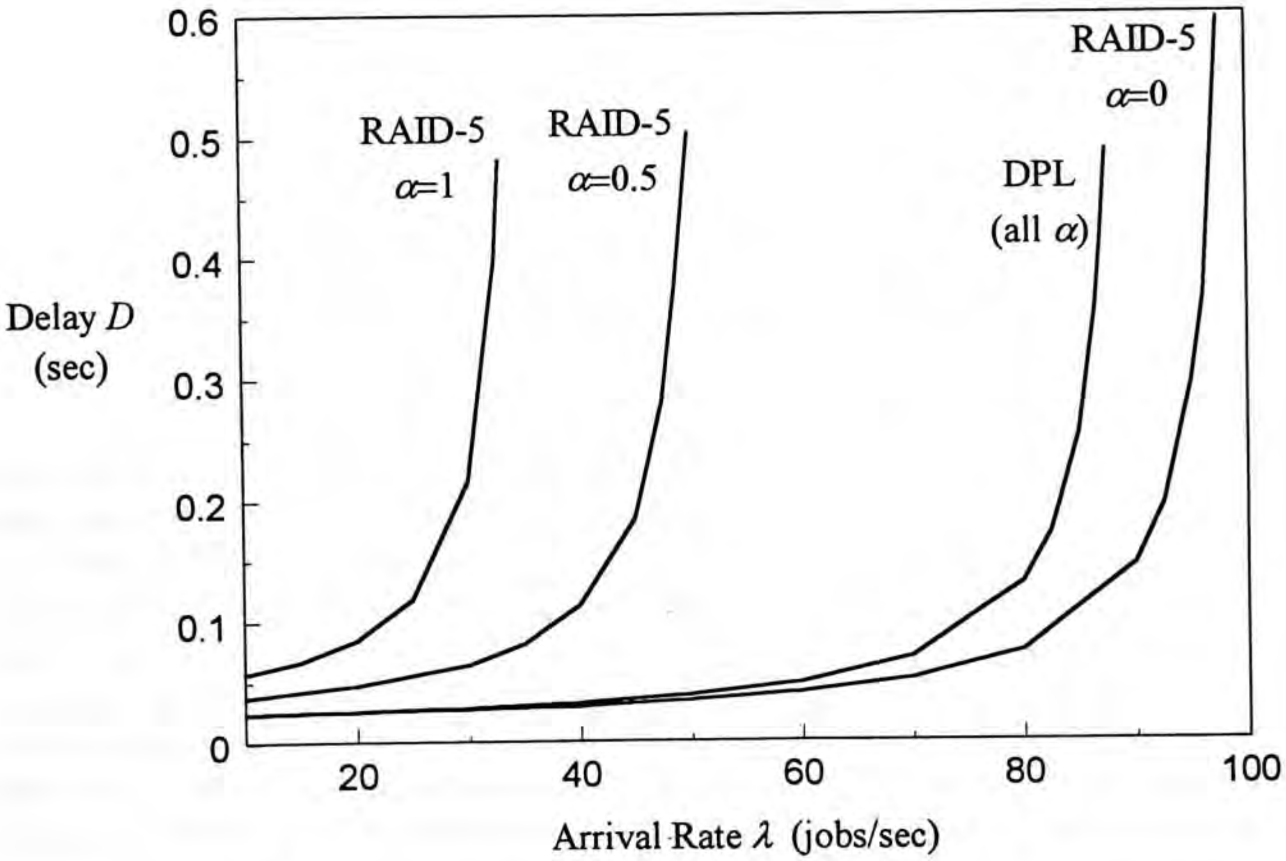


Figure 4-10 Delay througput characteristics of DPL Disk Array and RAID level 5.

D) Throughput Performance Using a Precise Disk Model

In our previous analysis disk service time is assumed to be exponentially distributed. This is usually not true for practical disk drives. To better understand the performance of DPL Disk Array and other RAID architectures in practice, we perform throughput simulation on different RAID architectures by using a precise disk model. In our simulation, disks are *not* assumed to be rotationally synchronized and their simulation parameters are summarized in Table 4-1. Each disk access involves a seek time, a latency and a data transfer time. We use the seek profile in [23], which states that the seek time T_{seek} (in mSec) is related to seek distance x (in number of cylinders) by:

$$T_{seek} = \begin{cases} 0 & \text{for } x = 0 \\ 0.4623\sqrt{x-1} + 0.0092(x-1) + 2 & \text{for } x > 0 \end{cases} \qquad (4\text{-}16)$$

Latency is assumed to be uniformly distributed. Data transfer time for one sector is equal to the disk revolution time divided by the number of sectors per track as given in Table 4-1. With that, the mean service time for all jobs is computed to be 20 ms. As stated in [24], this kind of disk modeling provides more than 94% accuracy when ignoring the disk caching effect. Since disk caching has little impact on "write" performance (which we are most interested in), we can thus assume that the system has no disk caching mechanism.

Cylinders per disk	1024
Tracks per cylinder	14
Sectors per track	48
Bytes per sector	512
Block Size	2 KB
Revolution time	13.3 ms
Single cylinder seek time	2 ms
Average seek time	13 ms
Max. data transfer rate	1.7 MB/s

Table 4-1 Disk parameters used in simulation.

Figure 4-11 shows the maximum throughput for DPL Disk Array and three other RAID architectures against α. The number of data disks for each architecture in this particular example is equal to 4. The total number of disks being used therefore equals 6, 8, 5, and 5 for DPL Disk Array, RAID level 1, RAID level 4, and RAID level 5 respectively. We denote the proportion of updated blocks in DPL Disk Array as β as shown in the figure. We observe that the performance of DPL Disk Array is quite insensitive to β. At most 10% drop in throughput is observed for all values of α and $\beta \leq 0.4$. We also observe from the figure that α has little impact on the performance of DPL Disk Array, but it significantly affects the performance of other three RAID Architectures. When α=0, RAID level 1 performs the best and provides twice the maximum throughput than that of the other three architectures. However, RAID level 1 requires the largest number of disks. When the proportion of "write" jobs α increases, DPL Disk Array provides nearly constant throughput value while the throughput values for RAID levels 1, 4, and 5 drop significantly. When all the jobs are of the "write" type, DPL Disk Array peforms the best and provides slightly higher maximum throughput than RAID level 1. From these results, we can conclude that DPL Disk Array provides the best "write" performance.

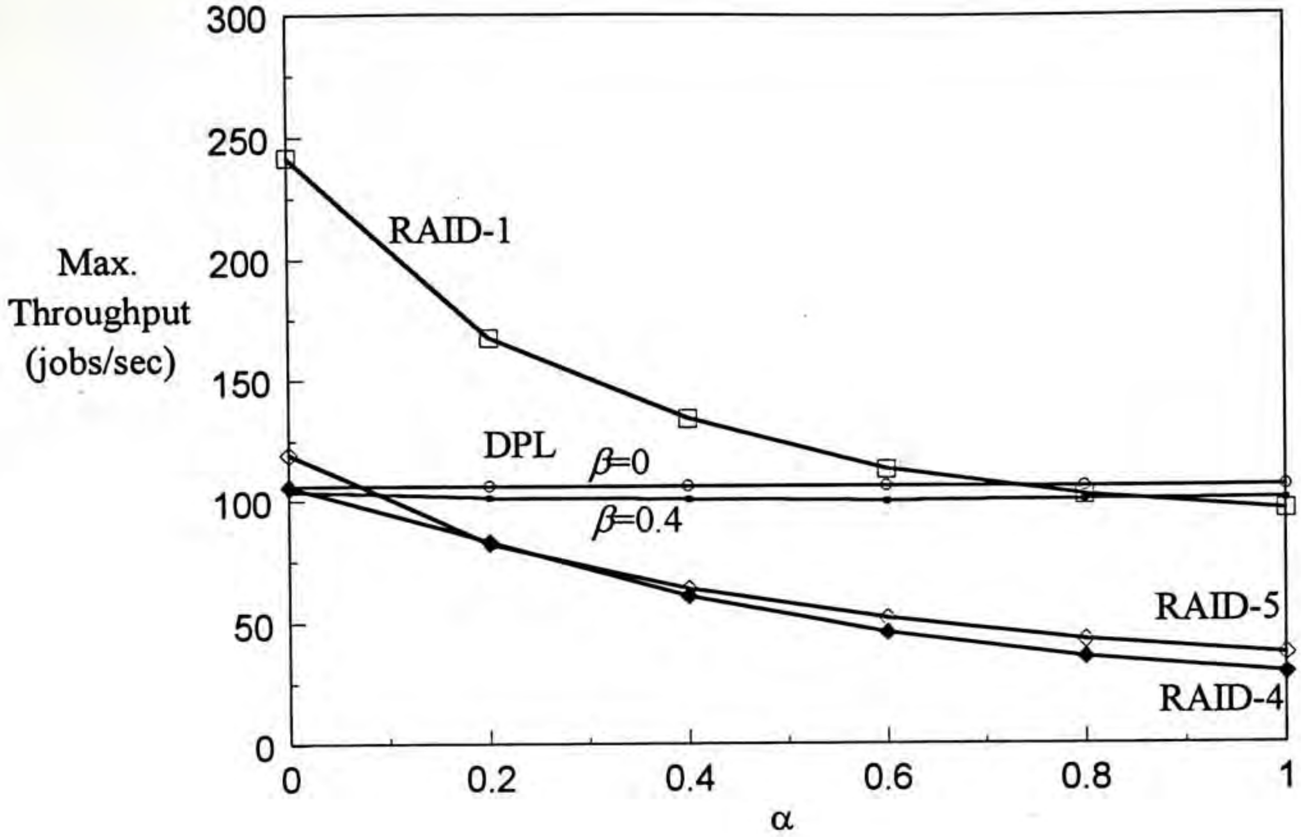


Figure 4-11 Maximum throughput comparison for various RAID architectures.

Figure 4-12 shows the maximum throughput per disk when all four architectures use the same total number of 24 disks. When α=0, we observe that RAID level 1 provides twice the throughput when compared to other three architectures. The trade-off for this good performance of RAID level 1 is the reduction of storage capacity by half. When the proportion of "write" jobs increases, we find that the throughput of DPL Disk Array remains constant while that for the other three architectures drop significantly. When $\alpha \geq$ 0.4, DPL Disk Array provides the highest maximum throughput per disk. Since "write" performance is critical to EDS, DPL Disk Array is well suiting for such kind of applications.

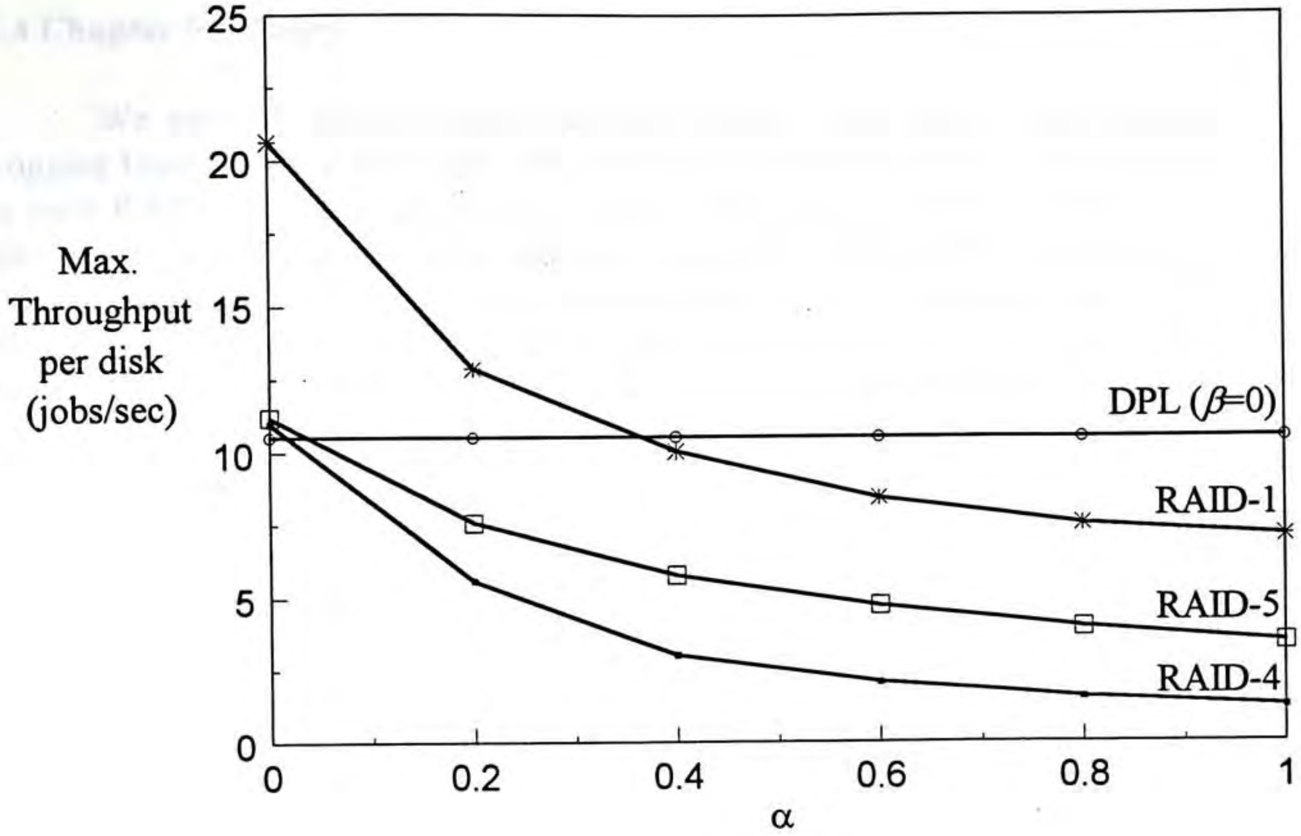


Figure 4-12 Maximum throughput comparison for various RAID architectures.

4.4 Chapter Summary

We propose in this chapter a new RAID architecture called Dynamic Parity Logging Disk Array for fast EDS. DPL Disk Array solves the small "write" problem found in most RAID levels and significantly reduces the blocking time for "write" transfers. It also has the journalling capability which is very desirable for EDS. Analytical results on DPL Disk Array shows that it provides much faster "write" response than RAID level 5. Throughput simulation using a precise disk model also shows that DPL Disk Array provides the highest "write" througput when compared to RAID levels 1, 4, and 5.

References

[1] Peter M Chen and David A Patterson, "Storage Performance - Metrics and Benchmarks," *Proceedings of the IEEE,* Vol. 81, No. 8, August 1993.

[2] Stanley Y W Su, *Database Computers,* McGraw-Hill, 1988.

[3] H Garcia-Molina, R J Lipton, and J Valdes, "A Massive Memory Machine," *IEEE Transactions on Computers,* Vol. C-33, pp. 391-399, May 1984.

[4] R Hagmann, "A Crash Recovery Scheme for a Memory-Resident Database System," *IEEE Transactions on Computers,* Vol. C-35, pp. 839-843, September 1986.

[5] K Salem and H Garcia-Molina, "System M: A Transaction Processing Testbed for Memory Resident Data," *IEEE Transactions on Knowledge Data Eng.*, Vol. 2, pp. 161-172, March 1990.

[6] T Bowen, G Gopal, G Herman, and J William Mansfield, "A Scale Database Architecture for Network Services," *IEEE Communications Magazine*, Vol. 29, pp. 52-59, January 1991.

[7] G Herman, G Gopal, K Lee, and A Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems," *Proceedings of the ACM SIGMOD Conference*, 1987, pp. 97-103.

[8] S Banerjee, V O K Li, and C Wang, "Distributed Database Systems in High-Speed Wide-Area Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 4, pp. 617-630, May 1993.

[9] D Patterson, G Gibson, and R Katz, "A Case for Redundent Arrays of Inexpensive Disks (RAID)," *Proceedings of the ACM SIGMOD Conference*, pp. 109-116, 1988.

[10] G A Gibson, *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*, MIT Press, 1992.

[11] J Gray, B Horst, and M Walker, "Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput," *Proceedings of the 16th Very Large Databases Conference*, pp. 148-161, Australia, 1990.

[12] D Stodolsky, M Hollan, W V Courtright II, and G Gibson, "Parity-Logging Disk Arrays," *ACM Transactions on Computer Systems*, Vol. 12, No. 3, pp. 206-235, August 1994.

[13] A Bhide and D Dias, *Raid Architecture for OLTP*, IBM Computer Science Research Report RC 17879, 1992.

[14] J Menon and J Kasson, "Methods for Improved Update Performance of Disk Arrays," *Proceedings of the Hawaii International Conference on System Sciences*, pp. 74-83, 1992.

[15] R H Katz, *Information Management for Engineering Design*, Springer-Verlag, 1985.

[16] G Gardarin and E Gelenbe, *New Applications of Data Bases*, Academic Press, 1984.

[17] J L Encarnacao and P C Lockemann, *Engineering Databases*, Springer-Verlag 1990.

[18] P C C Wang, *Advances in Engineering Data Handling*, Kluwer Academic Publishers, 1984.

[19] G Ariav and J Clifford, *New Directions For Database Systems*, Ablex Publishing Corp., 1986.

[20] D N Chorafas and S J Legg, *The Engineering Database*, Butterworths, 1988.

[21] T S Perry, *Technology 1995: Consumer Electronics, IEEE Spectrum*, Vol. 32, No. 1, pp. 40-43, January 1995.

[22] L Geppert, "Technology 1995: Solid State," *IEEE Spectrum*, Vol. 32, No. 1, pp. 35-39, January 1995.

[23] E K Lee and R H Katz, "The Performance of Parity Placements in Disk Arrays," *IEEE Transactions on Computers*, Vol.42, No.6, pp.651-664, June 1993.

[24] C Ruemmler and J Wilkes, "An Introduction to Disk Drive Modeling," *IEEE Computer Magazine*, Vol. 27, No. 3, March 1994.

Appendix

A) Pseudo Codes for System Backup Mode

```
process system_backup;
begin
/* backup data  disks */
for $i=1$ to $N$
    begin
    write $i$ and $c_i-1$ to tertiary storage;
    for $j=1$ to $c_i-1$
        begin
        write block $B(i,j)$ to tertiary storage;
        end;
    end;

/* backup parity disk */
write $c_p$ to tertiary storage;
for $j=1$ to $c_p$ do
    begin
    write block $B(N+1,j)$ to tertiary storage;
    mark block $B(N+1,j)$ as empty;
    end;
for $j=c_p+1$ to $K$
    begin
    mark block $B(N+1,j)$ as empty;
    end;

/* restore the most update contents of the popular blocks */
for $i=1$ to $N$
    begin
    for $j=K_D+1$ to $c_i-1$
        begin
        write block $B(i,j)$ to the location pointed by the block header of $B(i,j)$;
        mark block $B(i,j)$ as empty;
        end;
   for $j=c_i$ to $K$
        begin
        mark block $B(i,j)$ as empty;
        end;
    end;

/* clear the contents of the block location table */
for $i=1$ to $N$
```

```
  begin
  for j=1 to $K_D$
       begin
       s(B(i,j))=0;
       a(B(i,j))=0;
       end;
  end;

/* clear the contents of all counters */
for i=1 to N
   begin
   $c_i$=$K_D$+1;
   end;
$c_p$=1;
{switch the system to normal mode};
end;
```

B) Pseudo Codes for Normal Mode

```
process first_write_arrives;
/**** when the first "write" request arrives after each system backup *************
a request is a structure consists of: i) request type - req_type; ii) target block -
B(req_i,req_j); and for "write" requests: iii) new data - wri_data; iv) will be saved in
block - B(wri_i,wri_j); v) block parity set belonged to - par_set;  and vi) sequence
number in its parity set - block_no
i), ii) and iii) are supplied by the host, whereas the other parts are filled in by the I/O
processor before putting in the req_queue. req_queue is a FIFO queue of request's.
pb_image is a structure of parity block as shown in Figure 3, which consists of a
block header - header, and a parity sum - par_sum. image_queue is a FIFO queue
of pb_image's in the nonvolatile memory.
**********************************************************************/
begin
/* create the first parity set under construction */
parity_set_no=1;
no_in_set=1;
create a new pb_image in local memory;
request.B(wri_i,wri_j)=request.B(req_i,req_j);  /*write block to its original
location*/
write request.B(wri_i,wri_j) to pb_image.header;
write request.wri_data to pb_image.par_sum;
put pb_image to image_queue;

/* place the request to the request queue */
request.par_set=parity_set_no;
request.block_no=no_in_set;
```

```
no_in_set=no_in_set+1;
put request to req_queue;
end;

process request_arrives;
/**** when the requests other than the first "write" request arrives ****/
begin
/* a "read" request arrives */
 if request.req_type="read"
    begin
    put request to req_queue;
    end;

/* a "write" request arrives */
if request.req_type="write"
    begin
    /* test whether request.B(req_i,req_j) has been updated before */
    block_modified=no;
    if s(request.B(req_i,req_j))=1  /* modified bit=1=>the block has been updated*/
        begin
        block_modified=yes;
        end;
    if block_modified=no  /* if bit=0, check queue for updates on the same block */
        begin
        for each of the q_request in req_queue with q_request.req_type="write"
            begin
            if request.B(req_i,req_j)=q_request.B(req_i,req_j)
                begin
                block_modified=yes;
                end;
            end;
        end;

 /* if request.B(req_i,req_j) has been updated, choose a new location in PBA for the
 block */
 if block_modified=yes
    begin
    request.B(wri_i,wri_j)=B(wri_i,$c_i$);
    $c_i=c_i+1$;
    if $c_i=K+1$
        begin
        switch the system to system backup mode;
        end;
    end;
 else           /* if the block has not been updated, save to the original location */
```

```
    begin
    request.B(wri_i,wri_j)=request.B(req_i,req_j);
    end;

/* test whether the request can be assigned to the parity set in contruction */
append_to_parity_set=yes;
duplicate last image in image_queue to pb_image;
for each identifier B(wri_ii,wri_jj) in pb_image.header
    begin
    if wri_ii=wri_i
        begin
        append_to_parity_set=no;
        end;
    end;

/* yes, the request can be assigned to the parity set in construction */
if append_to_parity_set=yes and no_in_set ≤N
    begin
    pb_image.par_sum=pb_image.par_sum⊕request.wri_data;
    append identifier request.B(wri_i,wri_j) to pb_image.header as the last enrty of
            the header;
    put pb_image to image_queue;
    no_in_set=no_in_set+1;
    end;
else    /* no, the block cannot be assigned to the parity set in construction */
    begin    /* create a new parity set */
    parity_set_no=parity_set_no+1;
    no_in_set=1;
    create a new pb_image;
    write request.B(wri_i,wri_j) to pb_image.header;
    pb_image.par_sum=request.wri_data;
    put pb_image to image_queue;
    request.par_set=parity_set_no;
    request.block_no=no_in_set;
    no_in_set=no_in_set+1;
    put request to req_queue;
    end;
end;

process next_disk_operation;
/**** When a busy disk disk_i finishes its operation ****/
begin
if req_queue is empty    /* if the queue has no waiting requests */
    begin
    exit the process;    /* no operation is performed */
```

```
     end;

For the first request in req_queue which waits for disk_i, if any
     begin
     get request from req_queue;
     if request.req_type="read"    /* if the first request is of "read" type, */
         begin
         if a(request.B(req_i,req_j))=0    {0 address in block location table */
             begin
             read block request.B(req_i,req_j) from disk array;
             end;
         else    /* otherwise, the address is given by the entry in block location table */
             begin
             read block a(request.B(req_i,req_j)) from disk array;
             end;
         end;
         if operation_success=yes
             begin
             signal the host for "read" completion;
             exit from process;
             end;
        else
             begin
             switch the system to data recovery mode with fail_disk=disk_i;
             end;
      if request.req_type="write"    /* the request is of "write" type, */
         begin    /* write data block */
         write request.wri_data to request.B(wri_i,wri_j) in disk array;
         end;
      if operation_success=no
         begin
         switch the system to data recovery mode with fail_disk=disk_i;
         end;

      if request.block_no=1        /* write parity block, first data block of parity set? */
         begin
         $c_p$=$c_p$+1;    /* yes, write parity block of previous parity set to parity disks */
         if $c_p$=K+1    /* If parity disk is full */
             begin
             switch system to system backup mode;
             end;
         write pb_image to the parity disks at location pointed by $c_p$;
         end;
      get first pb_image from image_queue;  /* get parity image and write to parity
                                                                  disk*/
```

```
    if operation_success=no
        begin
        switch the system to data recovery mode with fail_disk=failed_par_disk
        end;
    /* "write" operation is successful, update the block location table */
    if s(request.B(req_i,req_j))=1     /* if the block has been updated before, */
        begin
        a((request.B(req_i,req_j))=request.B(wri_i,wri_j);     /* write new location */
        end;
    s(request.B(req_i,req_j))=1;     /* set modified bit of the block to "1" */
    signal the host for "write" completion;
    end;     /* request queue is not empty */
end;
```

C) Pseudo Codes for Data Recovery Mode

```
process recover_data_disk;
/**** when a data disk fail_disk fails, recover its data to spare disk N+3 ****/
/* restore old contents of last backup */
begin
i=fail_disk;
read $c_i$ of failed_disk from tertiary storage;
for j=1 to $c_i$
    begin
    read block B(i,j) from tertiary storage and save it to B(N+3,j);
    end;

/* mark unused blocks in PBA as empty */
for j=$c_i$+1 to K
    begin
    mark B(N+3,j) as empty;
    end;

/* restore most update contents of popular blocks at the last backup */
for j=$K_D$+1 to $c_i$
    begin
    write block B(N+3,j) to the location pointed by the header of B(N+3,j);
    mark block B(N+3,j) as empty;
    end;

/* recover the updated blocks in the last parity set in construction */
read pb_image from nonvolatile memory;
    if pb_image.header has a block identifier B(wri_i,wri_j) which wri_i=i
        begin
        B(N+3,wri_j)= parity sum of all blocks in this pb_image except
```

```
                                   B(wri_i,wri_j);
          end;
     end;

/* recover the updated blocks after last backup */
for j=1 to $c_p$
     begin
     read pb_image from B(N+1,j);
     if pb_image.header has a block identifier B(wri_i,wri_j) which wri_i=i
          begin
          B(N+3,wri_j)= parity sum of all blocks  in this pb_image except
                                   B(wri_i,wri_j);
          end;
     end;

/* resume to normal operations */
set disk N+3 to work as disk i;
switch system to normal mode and re-execute the interrupted disk operation;
end;

process recover_parity_disk;
/**** when a parity disk fail_disk fails ****/
begin
i=fail_disk;

/* set par_disk to the functioning parity disk */
if i=N+1
     begin
     par_disk=N+2;
     end;
else
     begin
     par_disk=N+1;
     end;

/* restore contents of parity disk to the spare disk N+3 */
for i=1 to K
     begin
     copy B(par_disk,i) to B(N+3,i);
     end;

/* resume to normal operations */
set disk N+3 to be the new parity disk;
switch system to normal mode and re-execute the interrupted disk operation;
end;
```

Chapter Five

Performance Analysis of Mirrored Disk Array

Previous performance studies on mirrored disk array are mainly by computer simulation or by approximate analyses which ignore the fork/join synchronization of the disks. In this chapter, an exact Markov Chain analysis of mirrored disk array is presented. The two disks are modeled as two independent exponential servers. Each "read" job is served by either one of the servers and each "write" job is forked into two independent sub-jobs for separate services in the servers. A "write" job is completed only when both sub-jobs are completed. The analysis is then verified by computer simulation.

5.1 Introduction

Mirrored disk array is one of the most common architecture for building reliable and fast I/O systems. It is defined as level 1 architecture of Redundant Arrays of Inexpensive Disks (RAID) [1]. In a mirrored disk array, data is duplicately stored into two identical disks for data reliability. No data is lost if any one of the disks fails. Mirroring the data on two disks also speeds up the I/O operations since simultaneous "read" operations can be performed. Mirrored disk array also has higher availability because the system can provide full service even when one mirrored disk fails. Because of this, mirrored disk array is considered the best I/O architecture for many applications.

As stated in [2], performance analysis of disk arrays is usually difficult due to the presence of queueing and fork/join synchronization. The difficulty is the same in analyzing the performance of mirrored disk array. A "write" request sent to a mirrored disk array is carbon copied or *forked* into two identical requests operated on the disks. The "write" request is completed only when both carbon copied requests are completed, or they must be *synchronized*. Due to this difficulty, performance studies on mirrored disk array are either done by simulation [3-4], or analyses which ignore the fork/join synchronization [5]. In [6], an analysis on the disk arm seeking for mirrored disk array is reported.

In this chapter, we present a Markov chain analysis on mirrored disk array. Analytical results are then compared to the simulation results to verify the analysis.

5.2 Queueing Model

Figure 5-1 shows the queueing model of the mirrored disk array. Job arrivals are assumed to be a Poisson process with rate λ. An arrived job is placed to the job queue waiting for service. The servers are independent and identical with FCFS service discipline. Let a job be of the "write" type with probability α and of the "read" type with the remaining probability. If a "read" job reaches the head of the queue, it can be served by either server. A completed "read" job then leaves the system immediately. For a "write" job, it is served only when both servers are idle. When both servers are idle, a "write" job is forked into two *sub-jobs* and be served by the two servers. All completed "write" sub-jobs enter the synchronization queue. They either merge with their associated sub-jobs and leave the queue immediately or wait there for the completion of the sub-jobs still in service. The service times of each "read" job and each "write" sub-job are assumed to be exponentially distributed with mean $1/\mu_r$ and $1/\mu_w$ respectively.

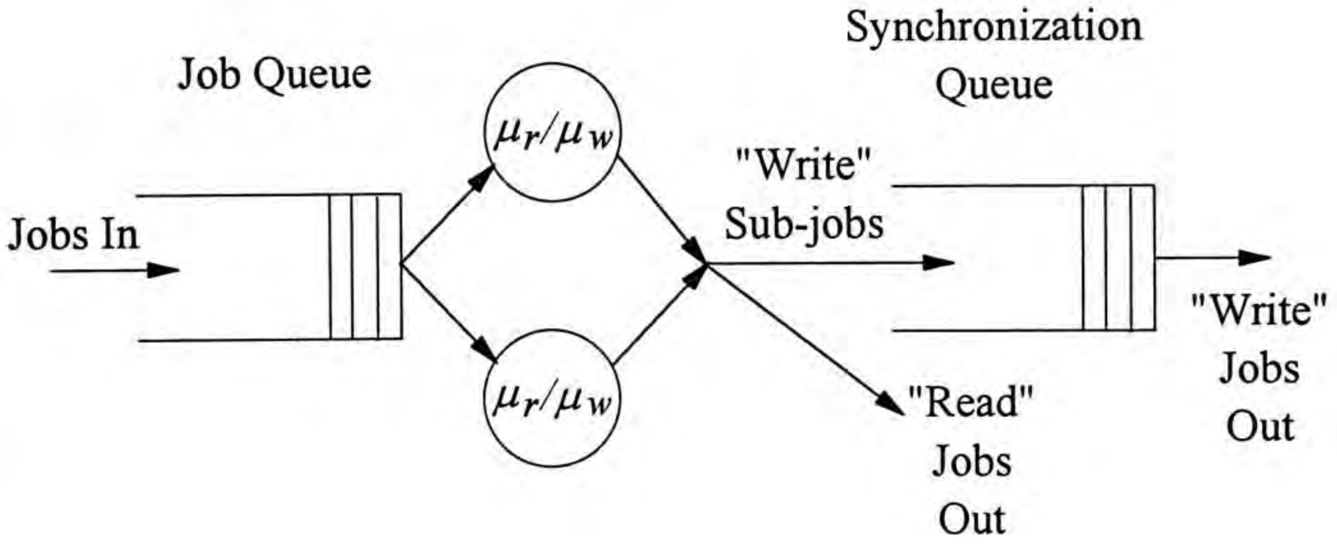


Figure 5-1 Queueing model of mirrored disk array.

5.3 Delay Analysis

Let $S_{s,q}$ denote the state of the system where q is the number of jobs in the queue and s is the status of the servers. The status depends on the job types currently being served and is tabulated as follow:

Status s	Description of status s
0	Both servers idle.
1	Only 1 server busy with a "read" job.
2	Both servers busy with "read" jobs.
3	Only 1 server busy with a "write" job.
4	Both servers busy with a "write" job.

We define the transition probability as

$$P[S_{s',q'}|S_{s,q}] = P\left[\begin{array}{l|l} \text{servers at status } s' \text{ and} & \text{servers at status } s \text{ and} \\ q' \text{ jobs in the queue} & q \text{ jobs in the queue} \\ \text{after state transition} & \text{before state transition} \end{array}\right] \qquad (5\text{-}1)$$

Consider a particular state transition. Define events E_1, E_2 and E_3 as:

E_1: A new job arrives.

E_2: A server finished serving a "read" job.

E_3: A server finished serving a "write" sub-job.

Let $v_{s,q}$ denotes the rate of state departure at $S_{s,q}$. It is given by

$$v_{s,q} = \begin{cases} \lambda & \text{for} s = 0 \\ \lambda + \mu_r & \text{for} s = 1 \\ \lambda + 2\mu_r & \text{for} s = 2 \\ \lambda + \mu_w & \text{for} s = 3 \\ \lambda + 2\mu_w & \text{for} s = 4 \end{cases} \qquad (5\text{-}2)$$

Therefore,

$$P[E_1]=\begin{cases}\dfrac{\lambda}{v_{s,q}} & \text{for} s=0\\ 0 & \text{for} s\neq 0\end{cases}$$

$$P[E_2]=\begin{cases}\dfrac{\mu_r}{v_{s,q}} & \text{for} s=1\\ \dfrac{2\mu_r}{v_{s,q}} & \text{for} s=2\\ 0 & \text{for} s\neq 1,2\end{cases}$$

$$P[E_3]=\begin{cases}\dfrac{\mu_w}{v_{s,q}} & \text{for} s=3\\ \dfrac{2\mu_w}{v_{s,q}} & \text{for} s=4\\ 0 & \text{for} s\neq 3,4\end{cases} \tag{5-3}$$

Table 5-1 shows the possible state transitions under each event E_i and the corresponding probabilities P[$S_{s',q'}$|$S_{s,q}$,E_i]. Description on each kind of state transition is given in the last column of the table. By removing the conditioning on E_i, the transition probabilities are obtained as:

$$P[S_{s',q'}|S_{s,q}]=\sum_{i=1}^{3}P[S_{s',q'}|S_{s,q},E_i]P[E_i] \tag{5-4}$$

Having obtained the transition probabilities, the set of state probabilities {P[$S_{s,q}$]} can be computed in the usual way. Then the long-term proportion of time spent in $S_{s,q}$, denoted as $p_{s,q}$, is given in [7] as:

$$p_{s,q}=\frac{P\left[S_{s,q}\right]/v_{s,q}}{\sum_{s'=0}^{4}\sum_{q'=0}^{\infty}P\left[S_{s',q'}\right]/v_{s',q'}} \tag{5-5}$$

Therefore, the expected numbers of jobs in the queue are given by:

$$E\left[N_q\right]=\sum_{q=0}^{\infty}q\left(\sum_{s=0}^{4}p_{s,q}\right) \tag{5-6}$$

Finally, by Little's formula, the expected sojourn time D of a job is given by

$$D=\frac{E[N_q]}{\lambda}+\left(\alpha\left(\frac{1}{2\mu_w}+\frac{1}{\mu_w}\right)+\frac{(1-\alpha)}{\mu_r}\right) \tag{5-7}$$

E_i	$S_{s,q}$	$S_{s',q'}$	P[$S_{s,q}$|$S_{s',q'}$, E_i]	Explanation on transition
E_1 {New job arrives}	$S_{0,0}$	$S_{1,0}$	1-α	A "read" job arrives and is served immediately.
		$S_{4,0}$	α	A "write" job arrives and is served immediately.
	$S_{1,0}$	$S_{1,1}$	α	A "write" job arrives and is blocked.
		$S_{2,0}$	(1-α)	A "read" job arrives and is served immediately.
	$S_{2,0}$	$S_{2,1}$	1	Blocking of new job due to no idle server.
	$S_{3,0}$	$S_{3,1}$	1	Blocking of new job due to the unfinished service to a "write" job.
	$S_{4,0}$	$S_{4,1}$	1	Blocking of new job due to the unfinished service to a "write" job.
	$S_{s,q}$, $s>0$, $q>0$	$S_{s,q+1}$	1	The new job enters an non-empty queue.
E_2 {"Read" job departs}	$S_{1,0}$	$S_{0,0}$	1	One "read" job departs.
	$S_{2,0}$	$S_{1,0}$	1	One "read" job departs.
	$S_{1,q}$, $q\geq1$	$S_{4,q-1}$	1	Job at the top of queue is a "write" job and is served.
	$S_{2,q}$, $q\geq1$	$S_{1,q}$	α	Job at the top of queue is found to be a "write" job and is served.
		$S_{2,q-1}$	(1-α)	Job at the top of queue is found to be a "read" job and is served.

Table 5-1 Possible state transitions given event E_i.

E_3 {"Write" job departs}	$S_{3,0}$	$S_{0,0}$	1	A "write" job departs.
	$S_{3,1}$	$S_{1,0}$	$1-\alpha$	The only job queued is found to be a "read" job and is served.
		$S_{4,0}$	α	The only job queued is found to be a "write" job and is served.
	$S_{3,q}$, $q>1$	$S_{1,q-1}$	$(1-\alpha)\alpha$	The job at the top of the queue is a "read" job and is served. The next following job in the queue is a "write" job is blocked.
		$S_{2,q-2}$	$(1-\alpha)^2$	Two jobs at the top of the queue are "read" jobs and are served. The next following job is blocked.
		$S_{4,0,q-1}$	α	The job at the top of the queue is a "write" job and is served.
	$S_{4,q}$, $\forall q$	$S_{3,q}$	1	The "write" job being served finishes using one of the servers.
All other state transitions			0	All transitions which are not listed above are invalid.

Table 5-1 Possible state transitions given event E_i (cont'd).

5.4 Numerical Examples and Simulation Results

Figure 5-2 shows the average job delay for a mirrored disk array under various traffic levels. We assume in this example that $\mu_r=\mu_w=50$ jobs/sec. Simulation on mirrored disk array was also performed to verify the analysis given above. We have extended the simulation time long enough so that the 95% confidence intervals is smaller than the size of the markers shown. From the figure we find that both the analytic and simulation results match very well with each other. We also find that when all the jobs are of the "read" type ($\alpha=0$), the maximum throughput for the mirrored disk array approaches 100 jobs/sec. This is expected because the system basically operates as an *M*/*M*/2 queue in this case. The maximum service rate of the system is just the total service rate of the two servers. When $\alpha=0.5$, we observe that the maximum throughput drops to about 43 jobs/sec. When all the jobs are of the "write" type ($\alpha=1$), the maximum throughput is about 33 jobs/sec.

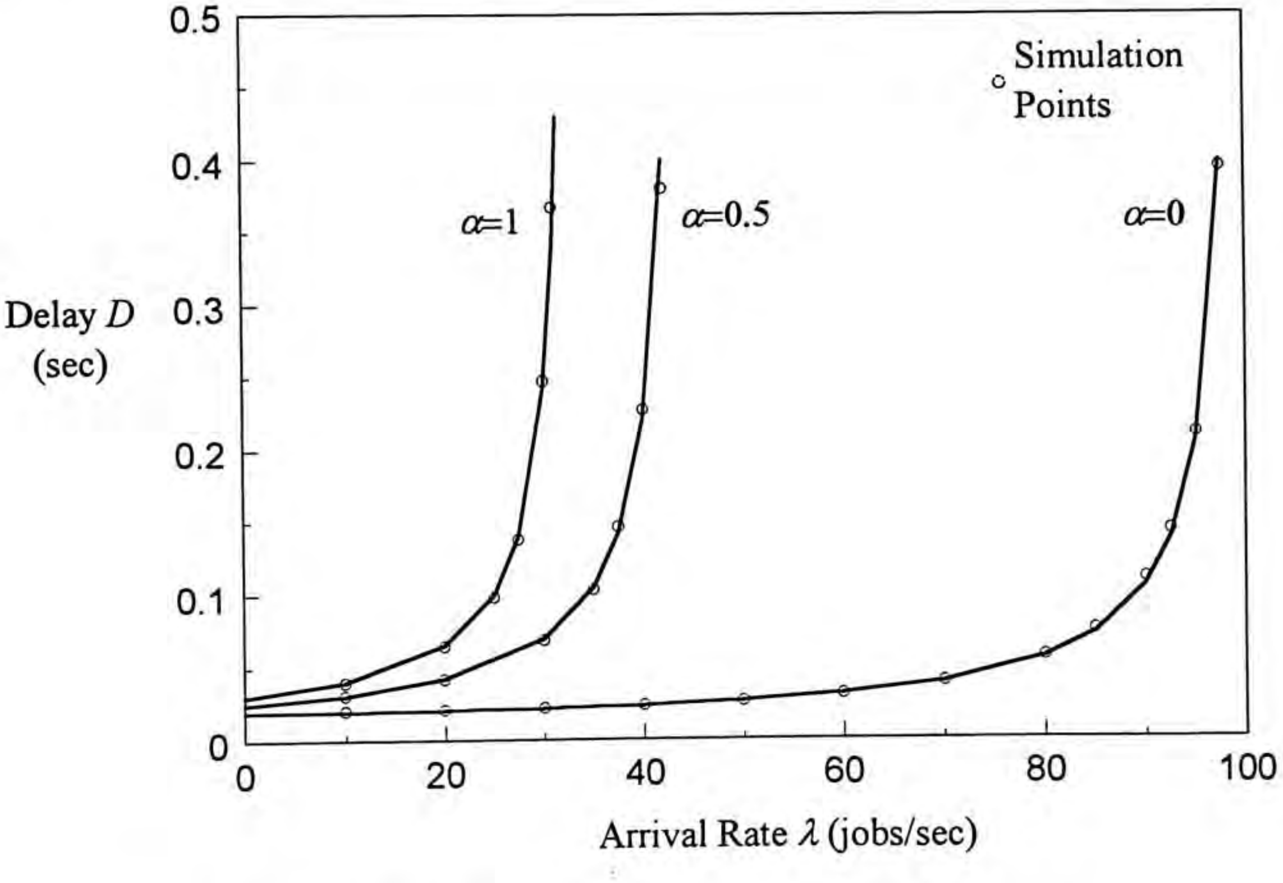


Figure 5-2 Delay throughput characteristics of mirrored disk array.

References

[1] D Patterson, G Gibson, and R Katz, "A Case for Redundent Arrays of Inexpensive Disks (RAID)," *Proceedings of the ACM SIGMOD Conference*, pp. 109-116, 1988.

[2] E K Lee and R H Katz, "An Analytic Performance Model of Disk Arrays," *Proc. ACM SIGMETRICS*, pp. 98-109, May 1993.

[3] C U Orji and J A Solworth, "Doubly Distorted Mirrors," *Proceedings of the ACM SIGMOD Conference*, pp. 307-316, Washington, USA, May 1993.

[4] Y Dishon and T S Liu, "Disk Dual Copy Methods and Their Performance," *Proc. of The 18th Int. Conference on Fault-Tolerant Computing*, pp. 314-319, 1988.

[5] S W Ng, "Improving Disk Performance Via Latency Reduction," *IEEE Trans. on Computers*, Vol. 40, No. 1, January 1991.

[6] D Bitton, "Disk Shadowing," *Proceedings of the 14th VLDB Conference*, Los Angeles, California, 1988.

[7] Alberto Leon-Garcia, *Probability and Random Processes for Electrical Engineering*, 2/Ed, pp. 485-6, Addison-Wesley, 1993.

Chapter Six

State Reduction in the Exact Analysis of Fork/Join Queueing Systems

A state reduction technique for the exact analysis of fork/join queueing systems is presented in this chapter. The technique is based on the standard Markov model and can be applied to systems having K homogeneous exponential servers. For a closed system with M jobs, the technique reduces the size of the state space from $(M+1)^K - M^K$ states to $\binom{M+K-1}{K-1}$ states. This amounts to more than five orders of magnitude of state reduction for a typical value of K=M=10. The state reduction technique can also be applied to the analysis of an open fork/join queueing system. It reduces the size of the state space from $(B+1)^K$ states to $\binom{B+K}{K}$ states where B is the maximum number of jobs allowed in the open queueing system. The state reduction amounts to more than six orders of magnitude for a typical value of K=10 and B=500.

6.1 Introduction

The fork/join queueing model is very useful in the performance study of parallel computing systems such as disk arrays and multiprocessor systems. For example, a computer running multiple I/O intensive processes has a disk array for fast I/O operations. The processes running on the computer queue on the I/O queue most of the time since they are I/O intensive. When a process get its turn for I/O operation, it accesses data which is striped across all the disks of the disk array. Having completed an I/O operation, the process reenters the I/O queue within a very short time and waits for the next I/O operation. To study the performance of the system, we can model the disk array as a closed fork/join queueing system.

A closed fork/join queueing model can also be used to study the performance of multiprocessor systems. Consider a computer with K processors running batch jobs. Jobs are queued in a job queue with a maximum number of M jobs. When the computer starts serving a job, the job is divided into K tasks running on the K different processors. A job completes when all its K tasks complete. When the system is fully loaded, a completed job will immediately trigger a new job arrival. The job queue is therefore always occupied with M jobs. Under full load condition, this system can be modeled as a closed fork/join queueing system as shown in Figure 6-1. Analysis on closed fork/join queues can provide performance insight on this multiprocessor computer.

The analysis on fork/join queues is usually difficult due to the presence of queueing and fork/join synchronization. Take disk arrays as an example, an I/O request is broken up or *forked* into K ($K \geq 1$) disk requests on K different disks. The I/O request is completed only when all K disk requests are completed, or they must be *synchronized*. Because of this, performance studies on disk arrays are either done by simulation or by analysis which ignores either queueing or fork/join synchronization. Many references on disk array performance studies can be found in [1].

Exact analysis on a closed K-server fork/join queue using standard Markov chain technique was proposed in [2]. But the amount of computations required grows exponentially with K. This calls for many approximate analysis of K-server fork/join queues [2-4][1]. In this chapter, we present a state reduction technique for the exact analysis on fork/join queues having homogeneous servers. In the next section, we first discuss the application of the state reduction technique on a closed fork/join queueing system. We then extend our discussion to open fork/join queueing systems in section 6.3. At last, we conclude the chapter in section 6.4.

[1] For a literature review on the analysis of K-server fork/join queues, see references in [2].

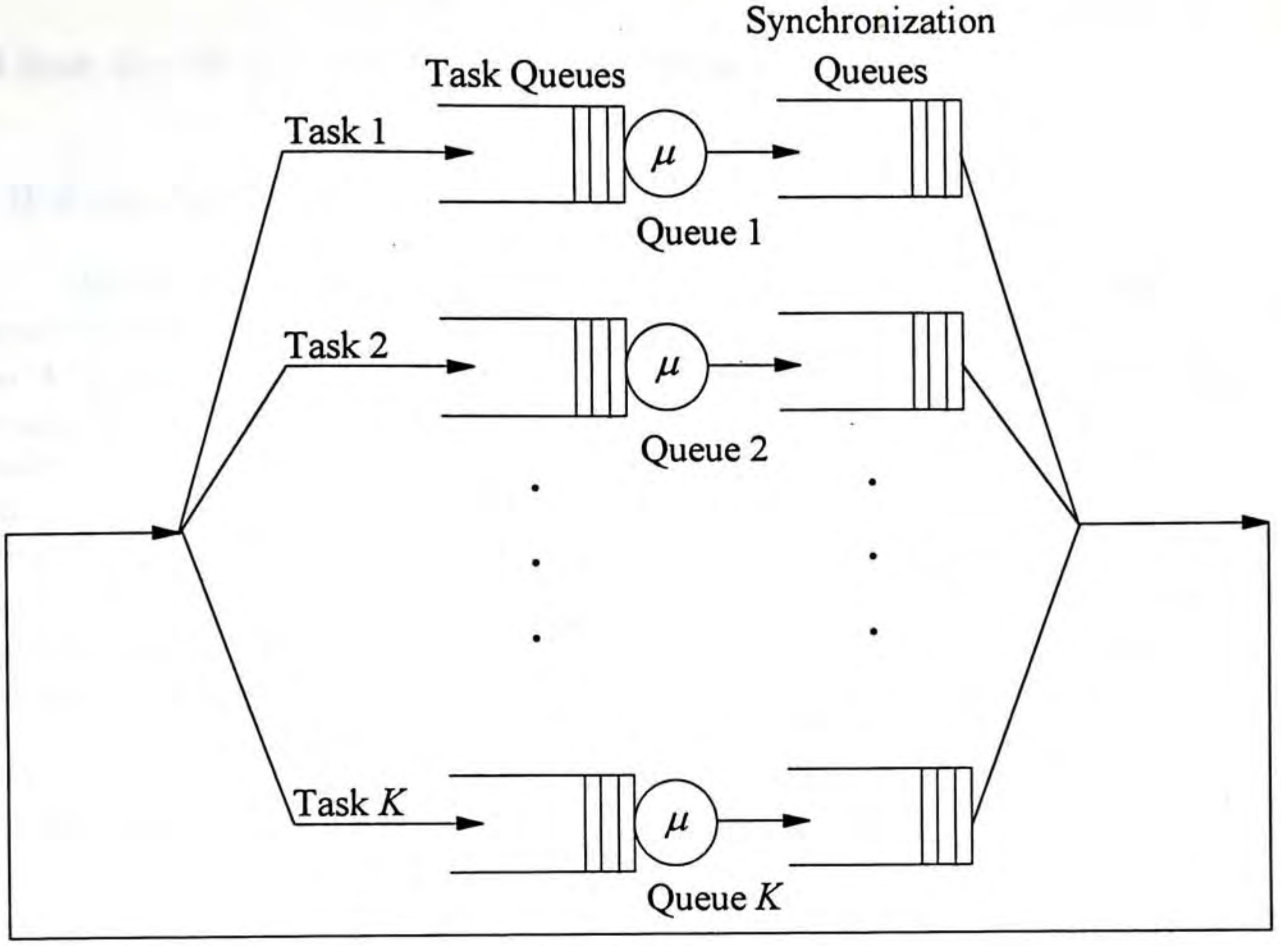


Figure 6-1 The closed fork/join queueing system.

6.2 State Reduction For Closed Fork/Join Queueing Systems

A. Queueing System Under Study

An M job closed fork/join queueing system is shown in Figure 6-1. When a job in the system departs, a new job immediately enters the system and *forks* into K tasks with task k (k=1,2,...,K) goes to the k^{th} *task queue*. The operations of the task queues are assumed to be independent. All tasks entering the same task queue are served in a FCFS manner. The service times for the tasks are assumed to be independent and exponentially distributed with mean $1/\mu$. Upon the completion of its service, a task enters the *synchronization queue* where it waits for the other tasks belonging to the same job. A job leaves the system only when all its K tasks are completed. The tasks belonging to a particular job are therefore *joined* before the job leaves the system. In the following, we present the state reduction technique and derive the average delay experienced by the jobs.

B. States and State Grouping

In conventional analysis the state is represented by the random vector $(N_1,N_2,...,N_K)$, where N_i denotes the number of task in task queue i including the one in service. When $N_1=n_1,N_2=n_2,...,N_K=n_K$, we denote the state of the system by $\underline{n}=(n_1,n_2,...,n_K)$. Since there are M jobs in the system, we have $\max(N_1,N_2,...,N_K)=M$. The number of possible states U for the system is given by:

$$\begin{aligned} U &= \begin{pmatrix} \text{The number of states with at least one queue having} \\ \text{occupancy } M \text{ and the remaining } K-1 \text{ queues having} \\ \text{occupancy } \leq M \end{pmatrix} \\ &= \begin{pmatrix} \text{The number of states} \\ \text{with all queues having} \\ \text{occupancy } \leq M. \end{pmatrix} - \begin{pmatrix} \text{The number of states} \\ \text{with all queues having} \\ \text{occupancy } \leq M-1. \end{pmatrix} \\ &= (M+1)^K - M^K \end{aligned} \tag{6-1}$$

Consider the arrival of a tagged job that brings the system to state $\underline{n}=(n_1,n_2,...,n_K)$. Let random variable X_i denote the delay experienced by task i of the new job. It is equal to the sum of n_i independent identically distributed exponential random variables and so has distribution

$$\text{Prob}[X_i \leq t] = \sum_{j=n_i}^{\infty} \frac{(\mu t)^j e^{-\mu t}}{j!} \tag{6-2}$$

Note that the average delays at tasks queues j and k are the same if $n_j=n_k$. The average delay X experienced by the tagged job is given by

$$X = \max(X_1, X_2, ..., X_K) \tag{6-3}$$

Since the servers are independent, the distribution of X is given by:

$$\text{Prob}[X \le t] = \text{Prob}[X_1 \le t]\text{Prob}[X_2 \le t]\ldots\text{Prob}[X_K \le t] \tag{6-4}$$

The task queues and the servers are all identical. Therefore any two jobs which begin at two different states, says $\underline{n}=(n_1,n_2,\ldots,n_K)$ and $\underline{n}'=(n'_1,n'_2,\ldots,n'_K)$, will have the same average delay if the states have the same *ranked* list of task queue lengths. We can thus merge all states having the same ranked list of queue lengths into a single system state. This is the basic idea of the state reduction technique.

C. Delay Analysis

Let random variable Q_i denote the number of *task queues* with length i where i includes the task in service. Since the system has K task queues, we have

$$\sum_{i=0}^{M} Q_i = K \tag{6-5}$$

Since there is at least one task queue which has M tasks, $Q_M \ge 1$. State occupancy time is the time between two successive task departures which is exponentially distributed. The evolution of $(Q_0,Q_1,\ldots,Q_M)$ is therefore a continuous time Markov Process. We denote system state as $\underline{q}=(q_0,q_1,\ldots,q_M)$ when $(Q_0,Q_1,\ldots,Q_M)=(q_0,q_1,\ldots,q_M)$.

Next, define the transition probability to be

$$\begin{aligned}&\text{Prob}[\underline{q}'|\underline{q}]\\ &= \text{Prob}[(Q_0,Q_1,\ldots,Q_M)=(q'_0,q'_1,\ldots,q'_M) \text{ after state transition}\\ &\qquad |(Q_0,Q_1,\ldots,Q_M)=(q_0,q_1,\ldots,q_M) \text{ before state transition}]\end{aligned} \tag{6-6}$$

When the system is at $\underline{q}=(q_0,q_1,\ldots,q_M)$, there are $q_1+q_2+\ldots+q_M$ non-empty queues. The rate of state departure at that state is therefore equal to $(q_1+q_2+\ldots+q_M)\mu$. The possible state transitions and the corresponding transition probabilities are calculated as follows:

$$\text{Prob}[\underline{q}'=(q_0,q_1,\ldots,q_{i-1}+1,q_i-1,\ldots,q_M)|\underline{q}] = \frac{q_i}{q_1+q_2+\ldots+q_M} \qquad 1 \le i \le M-1 \tag{6-7}$$

$$\text{Prob}[\underline{q}'=(q_0,q_1,\ldots,q_{M-1}+1,q_M-1)|\underline{q}] = \frac{q_M}{q_1+q_2+\ldots+q_M} \qquad q_M \ge 2 \tag{6-8}$$

$$\text{Prob}[\underline{q}'=(0,q_0,q_1,\ldots,q_{M-2},q_{M-1}+1)|\underline{q}] = \frac{1}{q_1+q_2+\ldots+q_M} \qquad q_M = 1 \tag{6-9}$$

Equations (6-7) and (6-8) represent all cases when no job arrival is triggered by a task departure. If the departed task is from a task queue with length i, the probability for this kind of state transitions is given by $q_i/(q_{1+\ldots+}q_M)$. This is because q_i task queues with length i is found before state transition out of a total of $q_{1+\ldots+}q_M$ non-empty queues. After state transition, $q'_i=q_i-1$ and $q'_{i-1}=q_{i-1}+1$ due to the task departure. Equation (6-9) represents the case when a task departure triggers one new job arrival. This kind of state transitions occurs when there is only one task queue with length M and a task departs from it. This departing task is the last task to finish for that particular job and so will trigger that job to depart from the synchronization queues immediately. As we are considering a closed queueing system, this means an immediate arrival of a new job to the task queues. Therefore the task queue with M tasks before transition will have $M-1+1=M$ tasks after state transition. In addition, the q_{M-1} task queues will each receive a task arrival to have $M-1+1=M$ tasks. Therefore, $q'_M=q_{M-1}+1$. For task queues with length i less than $M-1$, their lengths are increased by 1 after state transition. This make $q'_{i+1}=q_i$. Since all queues have at least 1 task after state transition, $q'_0=0$. The probability for this kind of state transitions is given by $q_M/(q_1+...+q_M)=1/(q_1+...+q_M)$.

Having obtained the transition probabilities, the set of equilibrium state probabilities {Prob[$\underline{q}$]} can be computed in the usual way. The long-term proportion of time spend in state $\underline{q}$, denoted as $p(\underline{q})$, is given in [5] as:

$$p(\underline{q})=\frac{(q_1+q_2+...+q_M)^{-1}\text{Prob}[\underline{q}]}{\sum_{\forall \underline{q}'\in S}(q'_1+q'_2+...+q'_M)^{-1}\text{Prob}[\underline{q}']} \tag{6-10}$$

where S is the set of all possible states and can be enumerated by a simple computer program.

The rate of task departures r_1 is :

$$r_1=\sum_{\forall \underline{q}\in S}p(\underline{q})(q_1+q_2+...+q_M)\mu \tag{6-11}$$

and so the rate of job departures r_2 is simply r_1/K. The job arrival rate is the same as the job departure rate for a closed queueing system. Using Little's formula, the expected job delay D in the fork/join queueing system is:

$$D=\frac{\text{Expected number of jobs in the system}}{\text{Job arrival rate}}=\frac{M}{r_2}=\frac{MK}{\mu\sum_{\forall \underline{q}\in S}p(\underline{q})(q_1+q_2+...+q_M)} \tag{6-12}$$

D. Computational Complexities

In the original state space, the total number of states is:

$$\begin{aligned}U&=(M+1)^K-M^K\\&=KM^{K-1}+\binom{K}{2}M^{K-2}+....+1\end{aligned} \tag{6-13}$$

If Gaussion Elimination is used to solve the state equations, the computation complexity

$$f_1(M,K)=O(U^3) \tag{6-14}$$

In the reduced state space, there is at least one task queue with length M. The remaining $K-1$ queues can have queue lengths ranging from 0 to M. The total number of queue length combinations can be found by comparing to the classical problem of finding the number of possible ways of distributing $K-1$ indistinguishable balls into $M+1$ urns. From [7], we find that the size of the new state space V is given by:

$$V=\binom{(K-1)+(M+1)-1}{K-1}=\binom{M+K-1}{K-1} \tag{6-15}$$

From Stirling's formula [8], we find that $n!$ can be approximated by:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \qquad (6\text{-}16)$$

Substitute into (6-15), we obtain

$$V \approx \sqrt{\frac{(M+K-1)}{2\pi M(K-1)}} \frac{(M+K-1)^{M+K-1}}{M^M (K-1)^{K-1}} \qquad (6\text{-}17)$$

Therefore, computation complexity of the reduced state space system $f_2(M,K)$ is simply:

$$f_2(M,K) = O(V^3) \qquad (6\text{-}18)$$

Note that (6-13) and (6-17) are complicated functions of M and K. To see how well the state reduction technique works, let us choose $K=M+1$ to obtain:

$$f_1 = O(M^{3M}) \qquad (6\text{-}19)$$

and

$$f_2 = O\left(\frac{2^{6M}}{M\sqrt{M}}\right) \qquad (6\text{-}20)$$

Therefore f_1 grows much faster than f_2. We will show some typical numbers in the next section.

E. Numerical Examples and Simulation Results

Figure 6-2 shows both analytic and simulation results of the average job delay D against the number of jobs M. We assume in this example that μ=1 for all servers. The simulation time is made sufficiently long to make the 95% confidence intervals smaller than the size of the markers shown. We find that for all values of K shown, D grows linearly with M.

Figure 6-3 compares the size of the original state space to that of the reduced state space for different values of M and K. Observe that when M=2 and K=4, state reduction is about one order of magnitude. When K increases to 10, state reduction reaches three orders of magnitude for the same M. When M is larger, we find that the state reduction is even more significant for all values of K shown. Comparing the two curves for M=10 at points K=10 we find that state reduction amounts to more than five orders of magnitude. The size of the original state space U in this case is about 1.5×10^{10} states, whereas the new state space has size $V\approx$90,000 states. We will not show any results on f_1 and f_2 as they are simply the cube of U and V respectively.

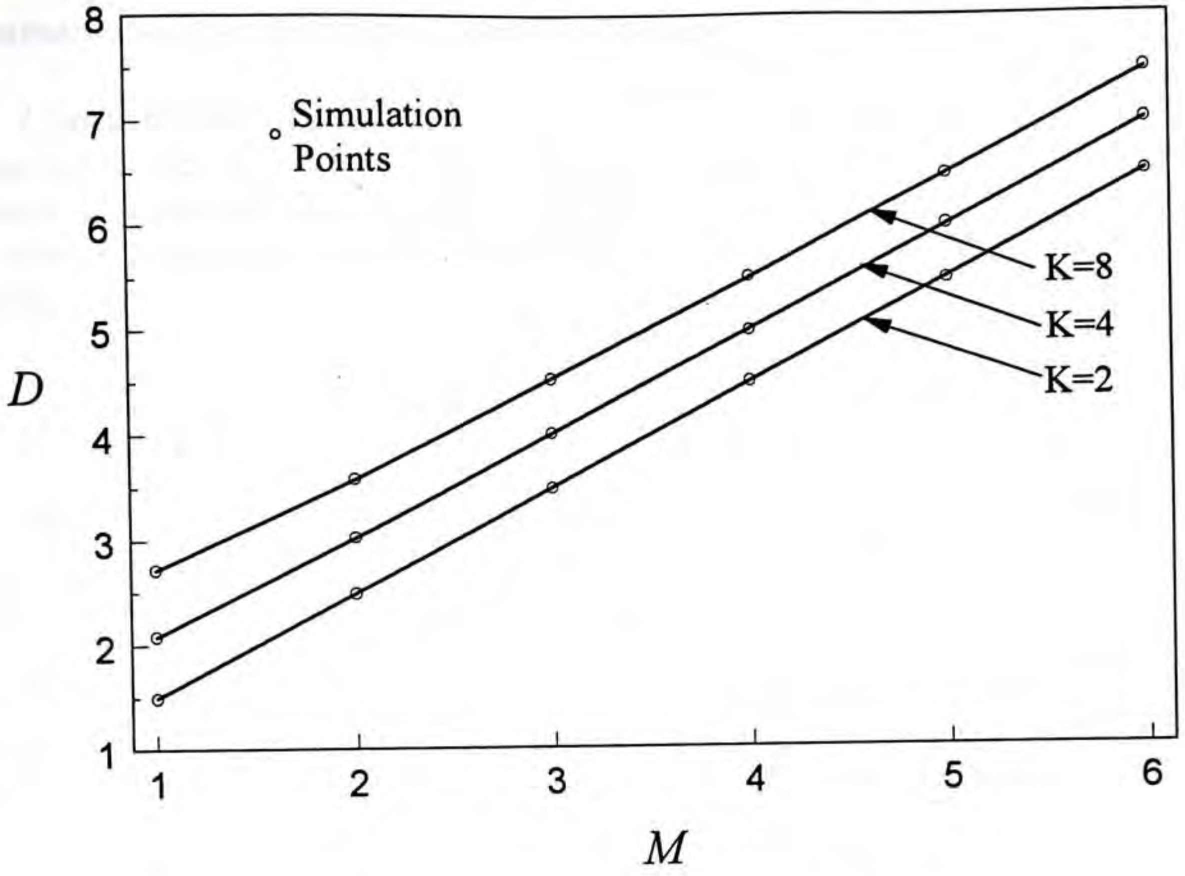


Figure 6-2 Average job delay against the number of jobs in the system.

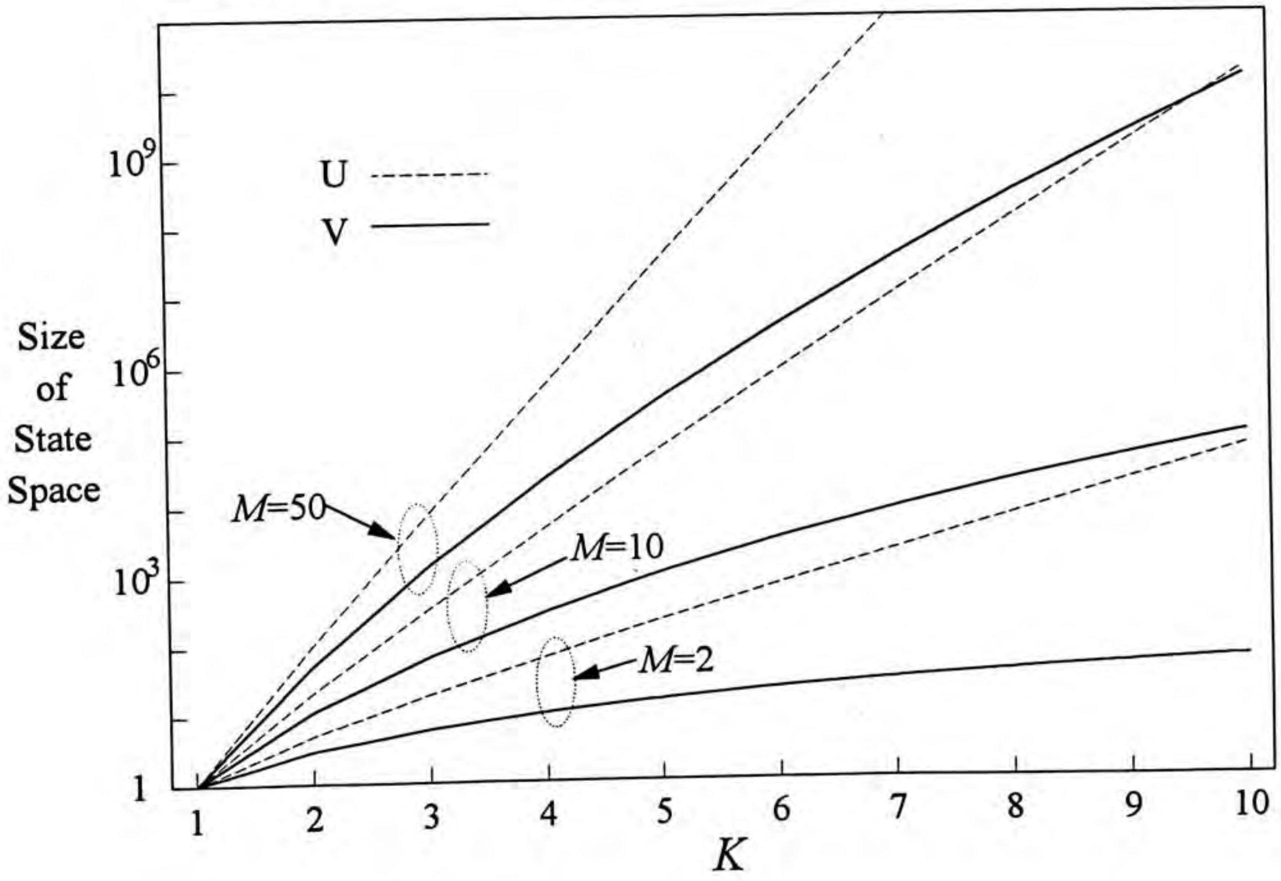


Figure 6-3 Comparison on the size of the original state space U and that of the reduced state space V.

6.3 Extension to Open Fork/Join Queueing Systems

Like the closed fork/join queueing model, the open fork/join queueing model is very useful in the performance study of many computer systems such as distributed replicated database and multiprocessor architecture [4]. In this section, we extend the state reduction technique described in section 6.2 to the analysis of open fork/join queueing systems.

A. Queueing System Under Study

The open queueing system under study is shown in Figure 6-4. Let the job arrivals be a Poisson process with rate λ. Upon arrival, a job is forked into K tasks with task k (k=1,2,...,K) being placed to k^{th} task queue. Tasks are served independently and in a FCFS manner. Upon the completion of its service, a task enters the synchronization queue where it waits for the other tasks of the same job. A job leaves the system only when all its K tasks are completed. Also let B be the maximum number of jobs allowed in the queueing system.

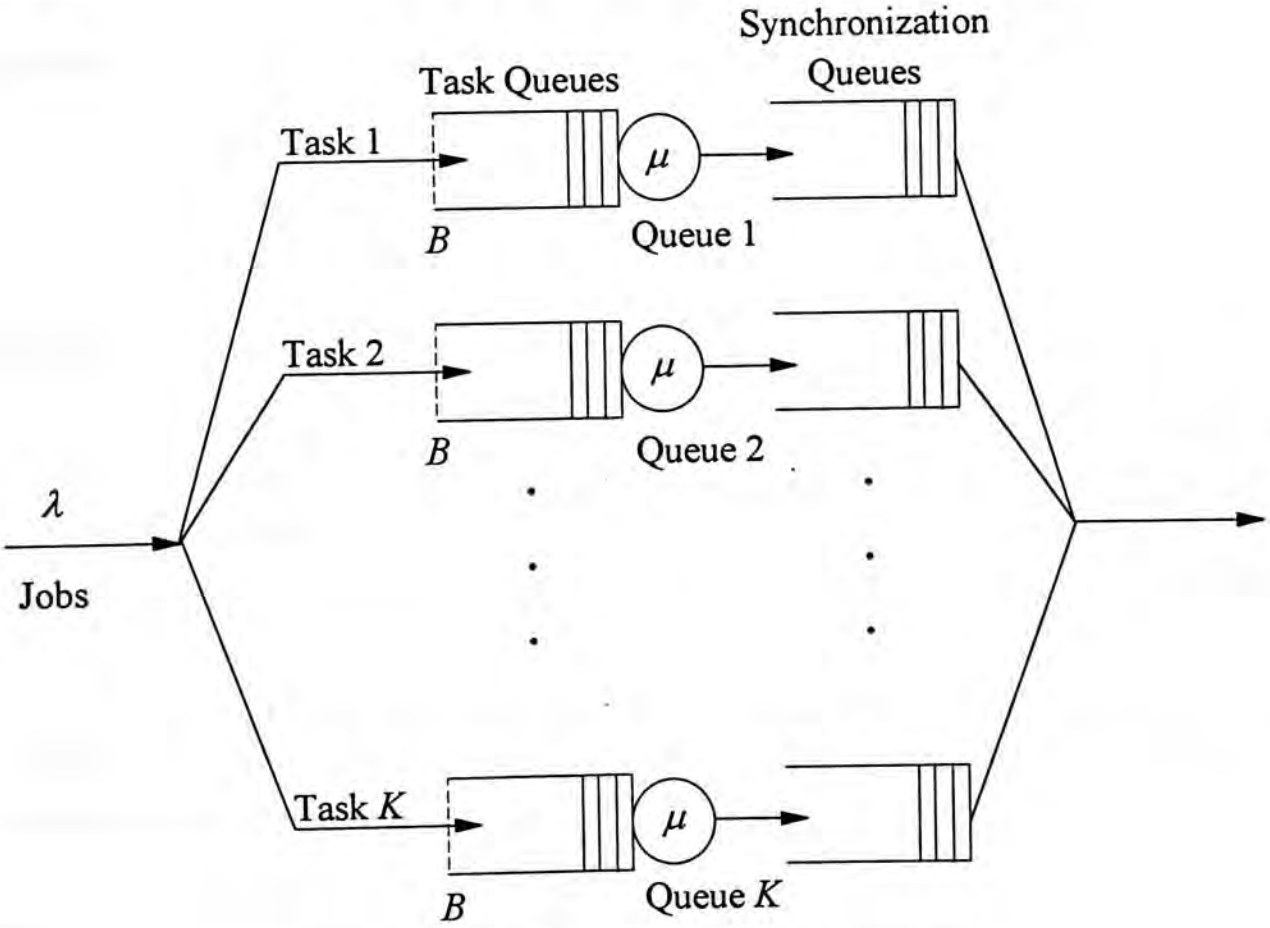


Figure 6-4 The open fork/join queueing system.

B. States and State Grouping

As before, let random variable Q_i denote the number of task queues with length i where i includes the task in service. Like the closed fork/join queueing system, the evolution of $(Q_0,Q_1,...,Q_B)$ is a Markov Process. We denote system state as $\underline{q}=(q_0,q_1,...,q_B)$ when $(Q_0,Q_1,...,Q_B)=(q_0,q_1,...,q_B)$.

C. Delay Analysis

Define the transition probability to be

$$\begin{aligned}&\text{Prob}\left[\underline{q}'\middle|\underline{q}\right]\\ &\quad=\text{Prob}\left[(Q_0,Q_1,...,Q_B)=(q'_0,q'_1,...,q'_B)\text{ after state transition}\right.\\ &\qquad\left.\middle|(Q_0,Q_1,...,Q_B)=(q_0,q_1,...,q_B)\text{ before state transition}\right]\end{aligned} \tag{6-21}$$

When the system is at $\underline{q}=(q_0,q_1,...,q_B)$, there are $q_1+q_2+...+q_B$ non-empty queues. Therefore, jobs arrives at rate λ and tasks departs with rate $(q_1+q_2+...+q_B)\mu$ at that state. The rate of state transition at $\underline{q}$ is therefore equal to $\lambda+(q_1+q_2+...+q_B)\mu$. Consider the departure of a task from task queue i. The probability that this event occurs before others is $q_i\mu/[\lambda+(q_1+q_2+...+q_B)\mu]$. After the departure, the number of queues with i tasks is $q'_i=q_i-1$ and the number of queues with $i-1$ tasks is $q'_{i-1}=q_{i-1}+1$. Therefore,

$$\text{Prob}\left[\underline{q}'=(q_0,q_1,...,q_{i-1}+1,q_i-1,...,q_B)\middle|\underline{q}\right]=\frac{q_i\mu}{\lambda+(q_1+q_2+...+q_B)\mu} \qquad 1\le i\le B \tag{6-22}$$

Consider the arrival of a job to the system. The probability that this event occurs before others is $\lambda/[\lambda+(q_1+q_2+...+q_B)\mu]$. After the job arrival, all queues increase their length by 1 and the number of queues with i tasks is $q'_i=q_{i-1}$. There is no more empty queue and $q'_0=0$. Since the maximum queue length is B, the number of queues with B tasks is $q'_B=q_B+q_{B-1}$. Therefore,

$$\text{Prob}\left[\underline{q}'=(0,q_0,q_1,...,q_{B-2},q_{B-1}+q_B)\middle|\underline{q}\right]=\frac{\lambda}{\lambda+(q_1+q_2+...+q_B)\mu} \tag{6-23}$$

Having obtained the transition probabilities, the set of equilibrium state probabilities $\{\text{Prob}[\underline{q}]\}$ can be computed in the usual way. The long-term proportion of time spend in state $\underline{q}$ is given in [5] as:

$$p(\underline{q})=\frac{[\lambda+(q_1+q_2+...+q_M)\mu]^{-1}\text{Prob}[\underline{q}]}{\sum\limits_{\forall\underline{q}'\in S}[\lambda+(q'_1+q'_2+...+q'_M)\mu]^{-1}\text{Prob}[\underline{q}']} \tag{6-24}$$

Next, let $L(\underline{q})$ be the length of the longest queue at state $\underline{q}$. It is equal to the number of jobs not yet completed at state $\underline{q}$. Therefore, the average number of jobs in the system, denoted as N, is given by:

$$N = \sum_{\forall \underline{q} \in S} L(\underline{q}) p(\underline{q}) \tag{6-25}$$

Using Little's formula, the expected job delay D in the fork/join queueing system is:

$$D = \frac{N}{\lambda} \tag{6-26}$$

D. Sizes of the State Spaces

In the original state space, the total number of states U is $(B+1)^K$ because all K queues can have queue lengths ranging from 0 to B. The total number of queue length combinations in the reduced state space can be found by comparing to the classical problem of finding the number of possible ways of distributing K indistinguishable balls into B+1 urns. From [7], we find that the size of the new state space V is given by:

$$V = \binom{K+(B+1)-1}{K} = \binom{K+B}{K} \tag{6-27}$$

If Gaussion Elimination is used to solve the state equations, the computation complexities for these two state spaces are similar to those of the closed fork/join queues discussed in Section 6.2.

E. Numerical Examples and Simulation Results

Figure 6-5 plots the average job delay against the arrival rate. The service rate is assumed to be 1 for all servers. The exact matching between the analytic and simulation results verify the analysis given above.

Figure 6-6 compares the size of the original state space to that of the reduced state space for open fork/join queueing systems. Observe that for both B=100 and B=500, the state reduction amounts to more than six orders of magnitude when K=10.

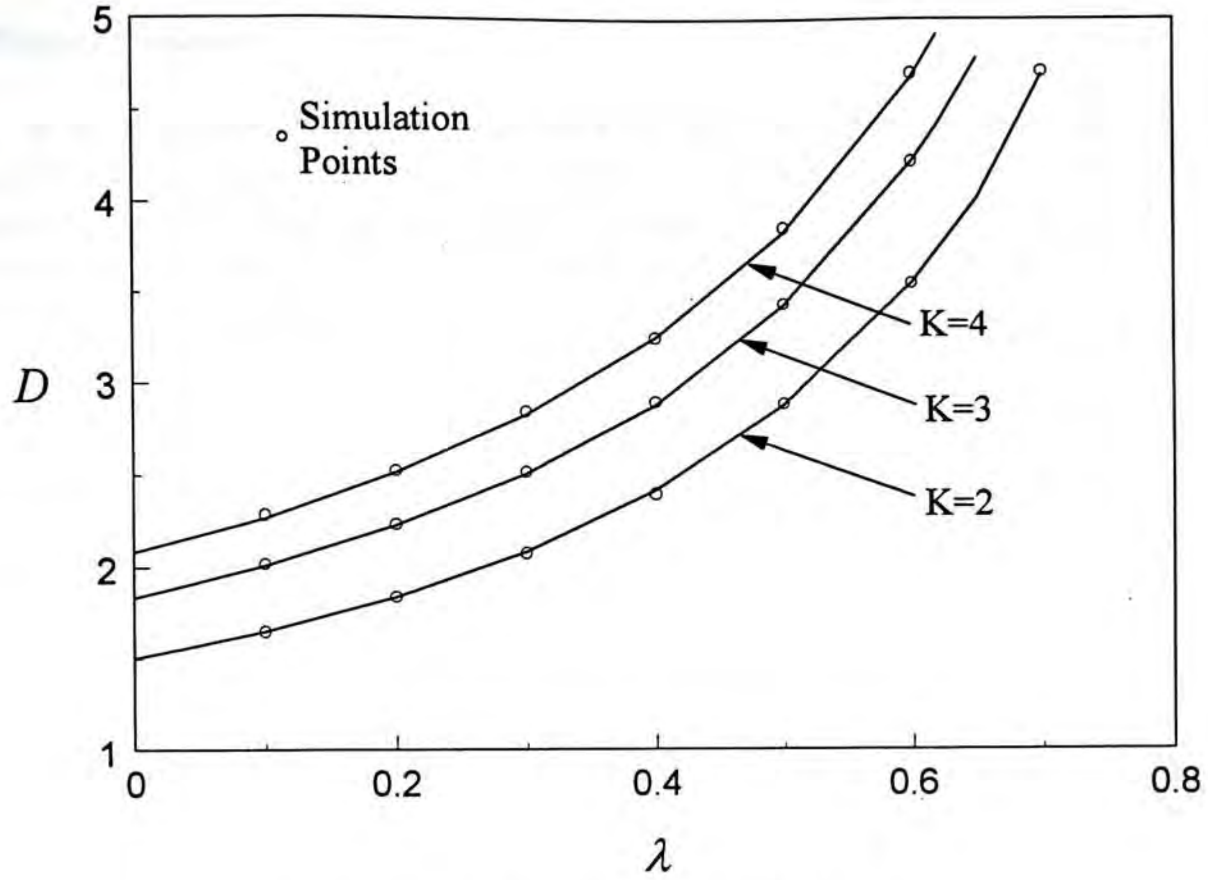


Figure 6-5 Average job delay against the arrival rate.

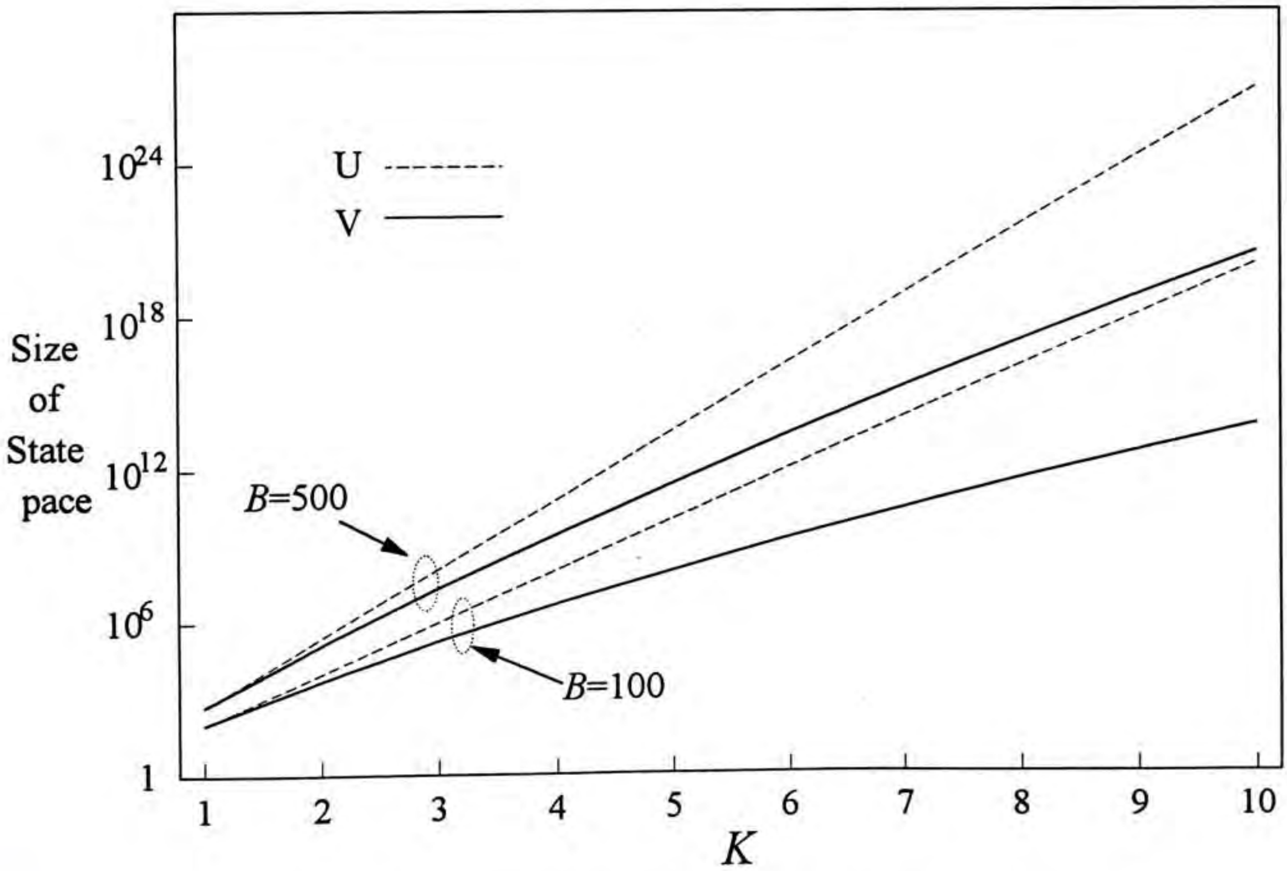


Figure 6-6 Comparison on the size of the original state space *U* and that of the reduced state space *V*.

6.4 Chapter Summary

A state reduction technique for the exact analysis of fork/join queues has been presented in this chapter. The technique is based on the standard Markov model and can be applied on a system having K homogeneous exponential servers. For a closed system with M jobs, the technique reduces the size of the state space from $(M+1)^K - M^K$ states to $\binom{M+K-1}{K-1}$ states. The state reduction amounts to more than five orders of magnitude for a typical value of K=M=10. For a open system, the technique reduces the size of the state space from $(B+1)^K$ states to $\binom{B+K}{K}$ states. The state reduction amounts to more than six orders of magnitude for a typical value of K=10 and B=500.

References

[1] Edward K. Lee and Randy H. Katz, "An Analytic Performance Model of Disk Arrays," *Proc. ACM SIGMETRICS*, pp.98-109, May 1993.

[2] Y. C. Liu and H. G. Perros, "A Decomposition Procedure for the Analysis of a Closed Fork/Join Queueing System," *IEEE Transactions on Computers*, Vol.40, No.3, March 1993.

[3] Philip Heidelberger and Kishor S. Trivedi, "Queueing Network Models for Parallel Processing with Asynchronous Tasks," *IEEE Transactions on Computers*, Vol.C-31, No.11, pp.1099-1109, November 1982.

[4] R. Nelson and A. N. Tantawi, "Approximate Analysis of Fork/Join Synchronization in Parallel Queues," *IEEE Transactions on Computers*, Vol.37, No.6, June 1988.

[5] Alberto Leon-Garcia, *Probability and Random Processes for Electrical Engineering*, 2/Ed, pp.485-6, Addison-Wesley, 1993.

[6] M. J. Maron and R. J. Lopez, *Numerical Analysis*, 3/Ed, pp.134-135, Wadsworth Publishing Company, 1991.

[7] Sheldon Ross, *A First Course in Probability*, 3/Ed, pp.14, Macmillan Publishing Company, 1989.

[8] M. O. Albertson and J. P. Hutchinson, *Discrete Mathematics with Algorithms*, pp.158, John Wiley & Sons, 1988.

Chapter Seven

Conclusion and Future Research

7.1 Summary

Previous studies on RAID show that using disk arrays can improve the I/O performance. The availability of commercial RAID products also indicates that disk arrays can practically be used in real systems. However, problems still exist in designing RAID systems for different kinds of applications. In this thesis, we propose techniques to solve some of the problems.

In chapter 1, we have briefly discussed why I/O system design is important in view of the rapid developments on computer and communication technologies. A brief introduction on Redundant Arrays of Inexpensive Disks (RAID) has also given.

In chapter 2, a novel technique called the **Selective Broadcast** technique for high speed data distributions is proposed. The technique significantly reduces the response time for data retrievals when compared to non-selective broadcast techniques. This chapter also gives a complete analysis on the Selective Broadcast technique.

In chapter 3, we address the problem of slow I/O rates for "write" operations found in RAID level 5. We propose a novel architecture called **Dynamic Multiple Parity (DMP) Disk Arrays**. DMP Disk Arrays reduces the blocking delays of "write" operations for database systems executing transaction in a strict order. Analysis on DMP Disk Arrays using Markov model is also given in the chapter.

Another disk array architecture called **Dynamic Parity Logging (DPL) Disk Arrays** is proposed in chapter 4 for fast engineering database systems. DPL Disk Arrays aim at both solving the small "write" problem found in RAID levels 4 and 5 and reducing the blocking delays for "write" operations. Analysis show that DPL Disk Array provide much higher "write" throughput than that of RAID level 5. DPL Disk Arrays also have the journalling capability which is very desirable for engineering database systems.

In chapter 5, a performance analysis on mirrored disk array is presented. The analysis is verified by computer simulation given in the chapter.

Chapter 6 describes a state reduction technique on the exact analysis on closed fork/join queues. Analysis on fork/join queues is very useful in the performance study on disk arrays. For typical values of system parameters, the proposed technique reduces the number of states required in describing the queueing systems by several orders of magnitude.

7.2 Future Research

Before the end of this thesis, we highlight some of the possible future research emerging from the work described above.

A. Selective Broadcast Technique

1. Based on the analysis given in chapter 2, we may study the average cycle time of disk arm movements for a disk using the CSCAN algorithm. The requests to the disk may be aperiodic.

2. We may extend the analysis given in chapter 2 to study the average cycle time of a token passing system. Input traffics to the nodes of the system may be asymmetric.

3. We may also study the cyclic behaviors of different kinds of polling systems by extending the analysis given in chapter 2.

4. We may apply the Selective Broadcast technique to specific systems such as the Video On Demand systems. The analysis given in chapter 2 can be modified to study the performance of those systems.

5. In the study of the optimal choice of block sizes, we have assumed that disk delay is a constant. Analysis with actual disk delays is desirable although it will be complicated.

B. DMP Disk Arrays

1. A global job queue for all disks is assumed in the analysis given in chapter 3. Although DMP Disk Arrays are best for database systems executing transactions serially, it is worthwhile to study DMP Disk Arrays having separate disk queues.

2. The analysis on DMP Disk Arrays may be extended to study other RAID architectures.

3. The reliability of DMP Disk Arrays should be studied.

C. DPL Disk Arrays

1. It is possible to construct *N*-dimensional DPL Disk Arrays. *N*-dimensional DPL Disk Arrays will survive under failure conditions of *N* simultaneous disks.

2. The reliability of DPL Disk Arrays should be studied.

3. Algorithms which can retrieve old contents of updated data should be developed.

4. The size of the block location table may be reduced by using appropriate techniques.

5. We may design another kind of DPL Disk Arrays which periodically write back contents of popular blocks to their original locations. The PBA for this kind of DPL Disk Arrays will therefore not overflow. Other than DBMS, such design is suitable for used in general applications which does not require to keep a long journal of data updates.

D. Performance Analysis of Mirrored Disk Array

1. The analysis can be extended to study the performance of other disk array architectures.

2. The analysis can be extended to study the performance of other computer systems such as multiple copy systems and distributed database systems.

E. State Reduction Technique

1. The state reduction technique and the analysis on fork/join queues may be applied to analyze different kinds of disk array architectures.

2. The analysis given in chapter 6 may be extended to study other types of computer systems such as data replication systems and distributed database systems.