디스크 미러링을 이용한 효율적인 복구 알고리즘

김 성 수† · 조 영 종†

요 약

본 논문에서는 미러링을 이용한 디스크 시스템의 복구기법을 제안하고 분석한다. 디스크 결함발생시 수행되어 지는 복구과정이 사용자 디스크 요구에 대한 평균 응답시간에 미치는 영향을 다룬다. 또한, 복구과정 및 성능에 미치는 응용의 입출력 패턴들의 효과를 분석한다. 실험 결과에 따르면 시스템의 부하가 커질수록 일양접근 패턴에 비해 비일양접근 패턴에 대한 평균 응답시간이 짧아진다. 시뮬레이션 결과는 짧은 지연을 가지는 제안된 복구기법이 기존의 기법보다 더 좋은 성능을 가짐을 보인다.

Efficient Repair Algorithms using Disk Mirroring

Sungsoo Kim † · Young-Jong Cho †

ABSTRACT

In this paper, we propose and analyze various repair methods for a disk subsystem using mirroring. We also study the effects of the repair process that is invoked on disk faults on the mean response time of user disk requests. Finally, we analyze the effects of two different access patterns (uniform and non-uniform) on the repair process and performance. According to the results, average response times for non-uniform access pattern compared with uniform access pattern become shorter as the system load increases. Our simulation results show that the proposed repair algorithm with a short delay gives a better performance than the previous algorithms.

1. Introduction

Performance of computer systems has improved rapidly in recent years. However, this improvement has not been even among all components of a computer system; some components, such as CPU (as measured by its clock speed) and memory size, have improved faster (increasing by 40-100% per year) while other components, such as I/O, have improved at a much lower rate (e.g., the improvement in seek time has been only 7% in the same time span) [1, 2]. Hence, the I/O subsystem has become more and more clearly the bottleneck of a computer system.

Disk I/O performance has been improved by enhancing the disk subsystems. Several studies have been reported in this area. The concept of *redundant arrays of inexpensive disks* (RAID) is one of the most popular approaches for disk arrays [3, 4, 5]. RAID utilizes models for different disk array algorithms, like mirroring, striping, and striping with parity. These algorithms provide an increase in either performance or reliability (or both).

※이 논문은 1996년도 아주대학교 연구비 지원에 의하여 연구되었음.

† 정 회 원: 아주대학교 정보통신대학 정보및컴퓨터공학부

논문접수: 1996년 9월 5일, 심사완료: 1997년 5월 19일

The combined analysis of performance and reliability (performability) has become one of the key issues in modern disk subsystems. The higher reliability usually causes a performance degradation in disk write requests because the same data must be stored in multiple locations. On the other hand, redundancy in the system is essential for a disk subsystem to survive media faults (i.e., problems in a disk unit to store or maintain data due to, for example, a deterioration of the magnetic media) [6, 7].

Generally, performance analysis of disk subsystems has been performed under steady state conditions. However, performance of a fault recovery method is important, especially when the disk subsystem is used in a real-time environment (where strict response time requirements must be met). For example, the mean response time is not sufficient to ensure that a disk subsystem can continue its operations for storing or retrieving data in a multimedia application such as audio or video playback and recording. Usually, performance is guaranteed only under either a steady or a degraded mode, but not while the system is under repair for a disk fault.

Generally, disk faults are permanent (not temporary) because a disk fault is mainly caused by a major problem with the disk. The disk repair process is based on using a spare disk that is utilized by copying the contents of an entire disk from the fault-free disk to the spare disk. RAID-0 using disk striping with 0% redundancy, has the lowest cost of any RAID organization. This approach has the best write performance, but it does not offer the best read performance. As reported by Chen, et al. in [8], redundancy policy like mirroring can give a better performance on reads due to its adaptive scheduling capability for requests on the disk with the shortest expected seek and rotational delays.

In this paper, performance estimates of a mirrored disk subsystem are made for different repair algorithms. The main objective of this paper is to study the performance of various repair scheduling algorithms during a disk subsystem recovery process.

The organization of this paper is as follows. Section 2 reviews briefly previous studies. In section 3, disk fault models are introduced. Section 4 discusses alternative repair algorithms. The performance of the repair processes is then analyzed in section 5. Finally, conclusions are presented in section 6.

2. Review

Usually, the main interest for performance evaluation has been concentrated on the steady state analysis of disk arrays [2, 9, 10, 11, 12]. Normal operations of disk arrays have been studied extensively. These studies have been concentrated on two major areas: reducing the response time [13, 14] and increasing the throughput of the disk subsystem [2, 9, 15].

The repair processing time is usually considered to be short compared with the normal operation time (i.e., the mean time between failures, MTBF, is much longer than the mean time to repair, MTTR) [16]. However, the effects of a repair process can be significant when tight requirements in response time are imperative (e.g., in a real-time system or multimedia storing/retrieving environments).

Two related studies on repair algorithms for disk arrays have been reported in [16, 17]. In [17], the system is modelled in three operational modes: normal, repair, and degraded modes. The normal operation mode describes the operations when all disk units are operable. Whenever a disk fault occurs and the repair process is not yet started, the system is said to be in a degraded mode. When the system is being repaired and some of the disks are not available, the system is considered to be in a repair mode. In this model, only a total disk failure is considered (i.e., sector or track faults are not considered). Also, the study of [17] has been restricted to a uniform access pattern. In [16], a comprehensive study of reliability of disk arrays has been made. This study has analyzed intensively the reliability of disk arrays as function of repair time.

However, it has not studied the implications of the repair process on performance.

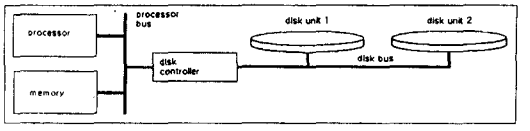
It has been shown that practical access patterns differ significantly from the uniform access pattern as the access probability is usually concentrated in certain locations [18]. Besides, the probability to access the same location (or nearby a previous disk request) may be substantial as indicated in [19].

3. Disk fault models

There are several disk array models of RAID [3, 4, 11, 12]. In this paper, only the mirrored disk system (RAID-l) is studied. A model of a mirrored disk subsystem is illustrated in Figure 1.

In this section, fault models of a disk subsystem are discussed. Only disk faults are considered. Faults in other parts of the computer system, such as disk controller, disk bus, memory or CPU, are ignored.

A fault in a disk unit can occur at any time during processing; however, the fault is generally detected only when the disk is used. Hence, a fault may be recognized only after a latency time, i.e. when a disk read or write request is made on the faulty area of the disk and this may occur at a substantial amount of time after the occurrence of the actual fault. A problem associated with fault latency is that it increases the probability of data loss (e.g., after a disk fault of one disk, the latent sector faults on the other disk result on data loss of those areas). Generally, it is assumed that the fault probability of the media is independent of the disk usage (i.e., the action of either reading or writing does not deteriorate the media).



(Fig. 1) Model of a mirrored disk system

An intensive study of disk fault models has been made in [6]. Disk faults can be divided into two main categories: sector and complete disk faults. Several subcategories can be used for defining a more detailed and accurate fault modeling of the disk subsystem. For simplicity, only disk faults are considered in this paper. Transient faults are processed as if they were permanent.

Different from a sector fault, a disk fault leads to a complete inoperability of the disk unit and a total data loss for the entire disk. Generally, disk faults are permanent (not temporary) because a disk fault is mainly caused by a major problem with the disk. The problem can be in the electric circuits (controlling, read/write, or interface logic) or mechanical components (such as seek of rotation motors). Also, a disk can be discarded due to an excessive sector fault rate or the exhaustion of spare sectors. In practical systems, these faults have been found to be permanent.

A disk fault can be detected in two ways. First, the fault can be detected when a disk stops responding to requests. Second, the disk can be treated initially for a sector fault but later it is recognized that actually the entire disk (or a significant part of it) is inaccessible and a sector fault is then diagnosed as a disk fault. In the first case, the detection of the fault is fast as the disk protocol detects missing disk units very efficiently. The second case may last longer because it is initially treated as a sector fault and some time is spent for diagnosing and correcting this fault.

The disk repair process is based on using a spare disk that is utilized by copying the contents of an entire disk from the fault-free disk to the spare disk. For an uninterruptible operation, the system should be provided with a hot standby disk.

The copying process can be simplified and expedited because the entire disk can be read and written sequentially. Hence, seek time can be minimized. The same process would be also applicable to rotation delays but, in this case, the main problem is that the number of sectors per track decreases as the track

number increases (disk tracks closer to the center have fewer sectors per track) and the actual physical disk structure is not generally known a priori. Usually, SCSI disks are represented as a continuous set of sectors and it is up to the implementation of the disk to organize the sectors (i.e., number of tracks, number of sectors per track, and number of disk surfaces).

One of the key issues in the repair of a disk system is when the repair process should be started. There are two major approaches for this problem: start the repair process immediately after fault detection, or delay the repair process until a more suitable time can be found (e.g., when there is no other load in the system).

In an immediate repair process, data is copied sequentially from a fault-free disk to a spare disk with no delay. A read request is issued to the fault-free disk. When the read request is completed, the data is written onto a spare disk. After the write request has completed, a new read request is issued immediately. This process is continued until all sectors are copied.

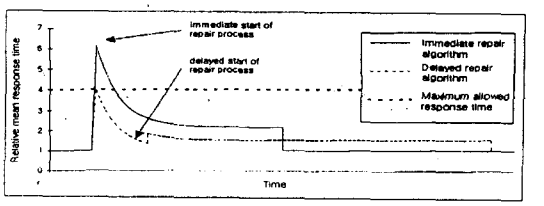
There are two major benefits from an immediate repair. First, it minimizes the probability of data loss (a fault on both disks in the same location at the same time). Second, the system regains sooner its normal performance level.

The disadvantage of an immediate repair algorithm is illusuated in Figure 2. A significant initial peak may result for the mean response time if the repair process starts immediately and the operations, which were originally allocated to the faulty disk, are redirected to the remaining (fault-free) disk. During the transient phase, the response time may exceed the limits imposed by system performance requirements. The duration of the initial peak is usually short because the number of requests waiting for disk access is generally small.

In a delayed repair algorithm, the peak in perfo-rmance degradation is lower than in the previous case (as illustrated in Figure 2) because the extended user requests and repair requests overlap only to a limited amount. Hence, it is easier to maintain response time requirements. By introducing a short delay between repair requests, it is possible to limit the number of repair read (or write) requests that any user disk request may need to wait.

However, there are two major disadvantages encountered by using this approach. First, the probability of losing data is higher as the time becomes longer, because only one disk is operable. Second, a longer rpair time causes a longer time of degraded performance prior to full availability for a normal performance level.

The fault probability of the second disk (while the first disk is under repair) is low because the repair time is significantly shorter than the mean time between failures, MTBF, of the disk [16]. Hence, the reliability of the disk system is not significantly decreased even if the repairing time is increased by a factor of two to four.



(Fig. 2) Example of the effects of immediate and delayed disk repair algorithms

4. Disk repair algorithms

Disk repair algorithms are divided into two major categories depending on the size of the fault, i.e., either individual sectors or the entire disk failure. Here, the interest is focused on repairing disk faults as sector faults can be repaired in a time that is in the order of a normal disk request while repairing a complete disk takes a significantly longer time.

The difference between disk repair algorithms is minor when it is detected by either a read and a write request. Only the beginning of the process is changed.

With a read request, the request is directed to the fault-free disk. A write request to the faulty disk is simply ignored because the operating system or the disk driver generates automatically a disk write request to both disks for a user write request.

The first disk repair algorithm is then given as follows.

Algorithm 1

[Step 1] Set the start address of the next disk request (denoted to as *sa*) to zero and set the request size (denoted to as *rs*) to a given value.

[Step 2] While the repair process is active, direct all read requests to the fault-free disk.
Direct a user write request to both disks (fauit-free and spare) if the user is requesting an address below *sa*, otherwise direct the write request only to the fault-free disk.

[Step 3] Wait for a given interval (denoted to as *wt*).

[Step 4] Issue a read request (with start address *sa* and size *rs*) to the fault-free disk.

[Step 5] When the read request is completed, issue a write request to the spare disk.
Increment the start address sa by the size of the requested block *rs*.

[Step 6] Wait for the write request to complete.

[Step 7] If there are still more sectors to be copied, go to step 2.

To maintain data integrity, two restrictions are imposed to the execution of Algorithm 1. First, disk requests should not be rearranged in the request queue. Rearrangement of disk requests could cause inconsistency in the disk storage. Second, step 5 should be completed as an atomic operation [20, 21]. If these conditions are not met, there is a possibility (e.g., a user write request is processed in the middle of step 5) that the spare disk is not updated properly (either a user write request is not written into the spare disk or the order of the write operations may be wrong). Another problem with Algorithm 1 is how to handle those user disk write requests that have the start and the end addresses at opposite sides of the parameter *sa*. The proposed approach splits these requests into two parts: the beginning (up to *sa*) is directed to both disks while the end (from *sa* to the end of the request) is directed only to the fault-free disk.

Algorithm 1 has two parameters that can be used for tuning up the repair process. First, the repair intensity (i.e., how long Algorithm 1 waits after completing the previous write to the spare disk before issuing the next read request to the fault-free disk) specifies the load of the repair process to the system. By delaying the next disk read operation, the effect of the repair process on user requests is reduced as the probability of having a pending repair request (i.e., when a user request enters the disk subsystem) is lower. Hence, the performance degradation of user disk requests is lower than when the repair process is continuously generating new repair requests. However, the slower repair process may cause the performance degradation to last a longer time.

Especially, the delayed repair process causes the fault-free disk to suffer from a heavier load because all read requests are directed to that disk. This can be reduced, provided read requests are divided similarly to write requests. However, these requests cause a longer processing time for repair requests in the spare disk (because read requests to the spare disk causes longer seek times for the repair writes). On the other hand, the fault-free disk would have fewer requests (especially those with longer seek times) and the load of the fault-free disk would be lower. Hence, it can be assumed that the response time of the entire disk subsystem should be better with a higher system load.

The second parameter of Algorithm 1 is the size of the request block. The larger the repair block is, the more it affects user requests because repair requests are then longer than conventional user requests. However, smaller repair blocks cause an I/O ove-

rhead because seek and rotation delays become significant (a significant part of the request processing is spent for seek and rotation, while the disk data transfer activity is low). A solution would be to access one (or several) complete track(s) per time. Then, the rotation and seek delays would be minimized. Unfortunately, there are two problems with this arrangement. First, conventional disk units have a variable number of sectors on a track (outer tracks have more sectors than inner tracks). Second, the exact internal structure of the disk is usually unknown.

A second disk repair algorithm is proposed; this algorithm is very similar to Algorithm 1 except for the handling of read requests.

Algorithm 2

[Step 1] Set the start address of the next disk request (sa) to zero and set the request size (rs) to a given value.

[Step 2] While the repair process is active, direct a read request to a fault-free disk if the start address is larger than (or equal to) sa or if the start address of the user request is on the second half of the disk. Otherwise, direct the read request to the spare disk. Direct a user write request to both disks (fault-free and spare) if the user is requesting an address below sa, otherwise direct the write request only to the fault-free disk.

[Step 3] Wait for a given time (wt).

[Step 4] Issue a read request (with start address sa and size rs) to the fault-free disk.

[Step 5] When the read request is completed, issue a write request to the spare disk.
Increment the start address sa by the size of the requested block rs.

[Step 6] Wait for the write request to complete.

[Step 7] If there are still more sectors to be copied, go to step 2.

The same restrictions (i.e., no rearrangements of requests and the atomic operation of step 5) as Algorithm 1 must be maintained. The benefit of Algorithm 2 compared with Algorithm 1 is the reduced load on the fault-free disk during the repair process. However, the load on the spare disk is increased because read requests tend to move the heads away from the next repair write location. Yet, this increase is not significant when there are also some write requests in the system (write requests below sa move the read/write heads of the spare disk away from the appropriate track of the next repair write).

5. Effects of repair algorithms

Parameters used in the evaluation are based on the model of [6]. These parameters are listed in Table 1.

〈Table 1〉 Numeric values for the simulation program

Component	Value	Unit	Comments
rwr	50/50	%/%	Read/Write ratio
t_{seek}	12	ms	End-to-end head movement time (excluding acceleration and deceleration time)
t_{acc}	6	ms	Acceleration and deceleration time for disk head
t_{tot}	1/60	s	Rotation time of the disk one full round
n_{track}	1024	tracks	Number of tracks on a disk
n_{disk}	512 x 1024	sector	Modelled disk size is 256MB

A simulation program is used for studying the performance of repairing an entire disk unit. Performance is studied by varying the user activity, and measuring the mean response time and percentiles of the distribution of user disk requests as well as the length of the repair process. Two different access patterns are used: uniform and non-uniform.

The simulation results indicate, that the response time is quite similar for uniform and non-uniform

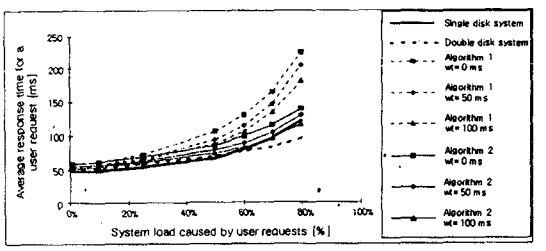
access patterns when the system load is moderate. With higher load, the non-uniform access pattern results in a better mean response time because the disk utilization is significantly different (requests concentrate on certain locations).

In Figure 3, the mean response time of user disk requests is presented as function of system load when the uniform access pattern is used. These results are compared also with the mean response time of a single disk (this represents the scenario when one disk is failed and repair has not yet started) and a mirrored disk system (this represents the scenario when the repair has been completed). With a moderate system load, the mean response time of user disk requests is slightly higher than the one with a single disk, but as the activity of user requests increases, the difference between them increases significantly, too. Especially Algorithm 1 suffers significantly from longer response times as it has a higher utilization of the fault-free disk. On the other hand, Algorithm 2 performs almost as well as a single disk (with no repair load) when a longer wait time ($wt = 100$ ms) is used. Algorithm 2 can even perform slightly better than a single disk because it can serve incoming user read requests using either of the disks (the amount of requests that the spare disk can handle depends on the current statistics of the repair process; i.e., how much of the disk is already copied). The single disk saturates earlier with heavy system loads.

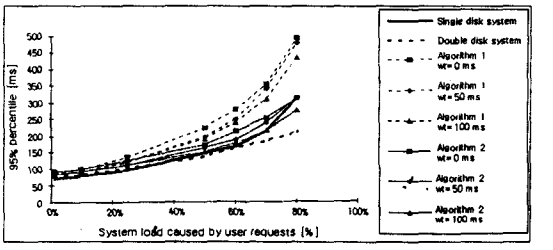
The average response time of user disk requests can be significantly reduced (15-20%) by using a small delay in the repair process. The most significant decrease in the response time is found in heavily loaded systems as the repair process is spread over a longer period of time, thus lowering the average system load.

Figure 3 also depicts that a single disk provides a slightly better response time than a mirrored disk when the system load is moderate. In a mirrored disk, a user write request suffers from longer rotation delays and longer seek delays (as read requests tend to move the read/write heads far from each other). The additional delays are compensated only in heavily loaded systems by the possibility to divide read requests to either of the disks (thus reducing the load of both disks).

The percentiles of the response time for a user request experience similar phenomena as the mean response time. The 95th percentile of the response time is illustrated in Figure 4 as function of the system load. The difference between Algorithm 1 and Algorithm 2 is more significant with the percentiles than with the mean response time. Also, the percentiles show that results with no and short delays ($wt = 50$ ms) give almost the same results (while the mean gives a significant difference). With a longer wait time ($wt = 100$ ms), the percentiles can be reduced. Besides, the single disk system results in higher percentiles than Algorithm 2 for a heavy system load.



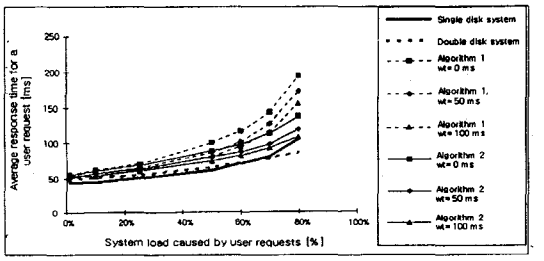
(Fig. 3) Mean response time of user requests as function of system load in a system with uniform access pattern



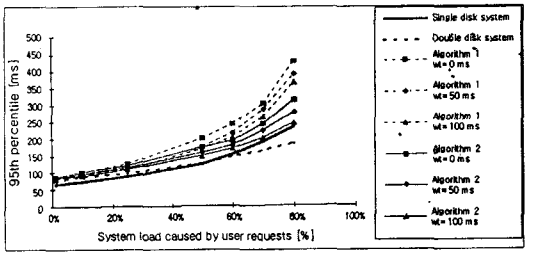
(Fig. 4) 95th percentile of the response time of user disk requests as function of system load in a system with uniform access patterns

In Figure 5, the mean response time of user disk requests is presented as function of system load when a non-uniform access pattern is used. Similar results as with a uniform access pattern but somewhat shorter response times are obtained. Also, the difference between Algorithm 1 and Algorithm 2 is smaller than with a uniform access pattern. This can be explained by the strong concentration of user disk requests on certain locations of the disk space, hence reducing the length of the average seek distance.

In Figure 6, the 95th percentile of the response time of user disk requests (using a non-uniform access pattern) is illusuated. By increasing the wait time (parameter *wt*) of the repair algorithms, the increase of the 95th percentile can be significantly reduced. A similar effect can be noticed also for the 90th and 99th percentiles.



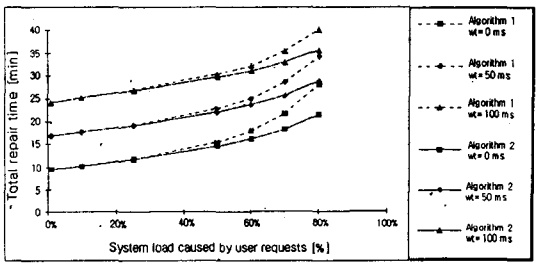
(Fig. 5) Mean response time of user disk requests as function of system load in a system with non-uniform access pattern



(Fig. 6) 95th percentile of the response time of user disk requests as function of system load in a system with non-uniform access pattern

The percentiles of the response time of user disk requests are very close to the ones of a system with a single disk (with no repair requests) when Algorithm 2 has a longer wait time ($wt = 100$ ms). Hence, the maximum response time of user disk requests can be kept moderate even during the disk repair process.

The cost for a smaller increase of the system response time consists of having a longer repair time. As illustrated in Figure 7, the repair time is significantly longer when a long wait time ($wt = 100$ ms) is used. When the system load is moderate or low and the uniform access pattern is used, the long wait time ($wt = 100$ ms) results in a total repair time 2.5 times longer than when zero wait time is used with both Algorithm 1 and Algorithm 2. With a non-uniform access pattern, the repair time is, respectively, up to 2.6 times longer. The repair process spends approximately 8(15) minutes of the total repair time in just waiting when the wait time is 50(100) ms while the actual repair takes about 9 minutes. The repair time is significantly shorter in Algorithm 2 with a heavy system load as it shares the load among the fault-free and spare disks more evenly than Algorithm 1.



(Fig. 7) Disk repair time as function of average user activity in a system with uniform access pattern

The longer repair time doesn't significantly reduce the system reliability. The repair time is still moderate even with long wait times (the repair time is less than 40 minutes when the system load is 80% and the wait time is 100ms). Hence, the reliability of the disk subsystem is not decreased much due to the longer

wait time as the MTBF of conventional disks is significantly longer (in order of tens of thousands of hours) [16]. At the same time, the effect on the response time of user disk requests is significant. Thus, the delayed repair can be used successfully in systems that have both high reliability and strict response time requirements.

6. Conclusions

In this paper, the performance effects of different repair algorithms for a mirrored disk subsystem (RAID-1) have been studied. A disk fault is handled using a spare disk to which data is copied from the fault-free disk. Two alternative repair algorithms have been studied. The difference of these algorithms is in the way they handle user read requests during the repair process. The simulation results indicate that it is more beneficial to share read requests among the fault-free and spare disks during the repair process than to let the fault-free disk to handle all user read requests. Especially, with higher system load, the difference between the results of the proposed algorithms is significant.

In the repair algorithms, a short delay between repair requests is introduced and studied. By having this short delay, it is possible to reduce significantly the performance degradation of user disk requests as measured by the mean response time and its percentiles. This delay increases the total repair time of the disk subsystem. However, its effect is very small as the repair time is still much shorter than the MTBF of the disks.

References

[1] P.M. Chen, G.A. Gibson, R.H. Katz and D.A. Patterson, "An Evaluation of Redundant Arrays of Disks using Amhdahl 5890," in Proc. of ACM Conference on Measurement & Modeling of Computer Systems, Vol. 18, No. 1, pp. 74-85, 1990.

[2] P.M. Chen and D.A. Patterson, "Maximizing Performance in a Striped Disk Array," in 17th Symp. Comp. Arch., pp. 322-331, May 1990.

[3] V. Catania, A. Puliafito, S. Riccobene and L. Vita, "Design and Performance Analysis of a Disk Array System," IEEE Trans. on Comp., Vol. 44, No. 10, pp. 1236-1247, Oct. 1995.

[4] A. Merchant and P.S. Yu, "Analytic Modeling and Comparisons of Striping Strategies for Replicated Disk Arrays," IEEE Trans. on Comp., Vol. 44, No. 3, pp. 419-433, Mar. 1995.

[5] M. Schulze, G. Gibson, R. Katz and D. Patterson, "How Reliable is a RAID?," in Proc. IEEE Spring Comp. Conf., San Francisco, Feb. 1989.

[6] H.H. Kari, H. Saikkonen and F. Lombardi, "Detection of Defective Media in Disks," in Proc. IEEE Int. Workshop on Defect and Fault Tolerance in VLSI Systems, pp. 49-55, 1993.

[7] S. Kim, *et al.*, "Repair Algorithms for Mirrored Disk Systems," in Proc. IEEE Int. Workshop on Defect and Fault Tolerance in VLSI Systems, Lafayette, LA, pp. 216-224, Nov. 1995.

[8] P. M. Chen, *et al.*, "RAID: High-Performance, Reliable Secondary Storage," ACM Computing Surveys, Vol. 26, No. 2, pp. 145-185, 1994.

[9] S. Stewart, "High Performance QIC Drives Increase Capacity and Data Rate," Computer Technology Review, pp. 57-62, May 1992.

[10] S.W. Ng, "Improving Disk Performance Via Latency Reduction," IEEE Trans. on Comp., Vol. 40, No. 1, pp. 22-30, Jan. 1991.

[11] D.A. Patterson, G. Gibson and R.H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," in ACM SIGMOD Conf., Chicago, pp. 109-116, May 1988.

[12] D.A. Patterson, G. Gibson and R.H. Katz, "Introduction to Redundant Arrays of Inexpensive Disks (RAID)," in IEEE Proc. COMPCON, Spring 1989.

[13] T.M. Olson, "Disk Array Performance in a Ran-

dom IO Environment," Computer Architecture News, pp. 71-77, Aug. 1989.

[14] M.Y. Kim, "Synchronized Disk Interleaving," IEEE Trans. on Comp., Vol. C-35, No. 11, pp. 978-988, Nov. 1986.

[15] A.L.N. Reddy and P. Banejee, "A Study of Parallel Disk Organizations," Computer Architecture News, pp. 40-47, Aug. 1989.

[16] G.A. Gibson, "Redundant Disk Arrays: Reliable, Parallel Secondary Storage," University of California TR UCB/CSD 90/615, Berkeley, Ph. D. Dissertation, Mar. 1991.

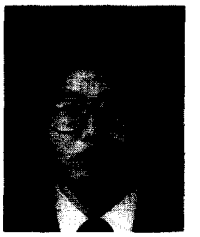
[17] J. Menon and D. Mattson, "Performance of Disk Arrays in Transaction Processing Environments," IBM Research Report RJ 8230(75424), p. 49, July 1991.

[18] J. Stockard, "I/O Workloads," in VII Data Storage Interface and Technology Conf., Technology Forums, pp. 20, Mar. 1991.

[19] M.Y. Kim and A.N. Tantawi, "Asynchronous Disk Interleaving: Approximating Access Delays," IEEE Trans. on Comp., Vol. 40, No. 7, pp. 801-810, July 1991.

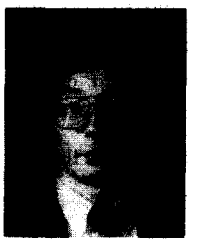
[20] C.J. Date, 'A guide to the SQL Standard', Addison-Wesley, New York, 1989.

[21] J. Gray, 'The Benchmark Handbook for Database and Transaction Processing Systems', Morgan Kaufmann Publishers Inc., San Mateo, 1991.



김 성 수

1982년 서강대학교 전자공학과(공학사)
1984년 서강대학교 전자공학과(공학석사)
1995년 Texas A&M University, 전산학과(공학박사)
1983년~1986년 삼성전자(주) 종합연구소 컴퓨터연구실(주임연구원)
1986년~1996년 삼성종합기술원 수석연구원
1991년~1992년 Texas Transportation Institute 연구원
1993년~1995년 Texas A&M University, 전산학과, T.A.
1996년~현재 아주대학교 정보통신대학 정보및컴퓨터공학부 조교수
1997년~현재 한국 정보처리학회, 한국 정보과학회 논문지 편집위원
관심분야: 결함 허용, 멀티미디어 시스템, 이동 컴퓨팅, VLSI, 성능평가 등



조 영 종

1983년 서울대학교 전자공학과(공학사)
1985년 한국과학기술원 전기및 전자공학과(공학석사)
1989년 한국과학기술원 전기및 전자공학과(공학박사)
1985년~1988년 금성전기연구소 네트워크연구실(주임연구원)
1989년~1995년 LG정보통신 중앙연구소 ATM 교환연구실(책임연구원)
1991년 AT&T Bell Lab 파견연구원
1996년~현재 아주대학교 정보및컴퓨터공학부 조교수
관심분야: 컴퓨터네트워크, 초고속통신망, 무선통신망, 무선ATM, 성능평가 등