적응형 패리티 그룹 구성을 이용한 RAID 5 제어기에서의 캐시 운영

허 정 호[†] · 송 자 영[††] · 장 태 무[†††]

요 약

RAID 5는 고 신뢰도 및 고성능 디스크 시스템을 구성하는 널리 사용되는 기법이다. 본 논문은 특히 OLTP(On-Line Transaction Processing System) 작업환경에서 RAID 5의 소규모 쓰기("small write") 문제를 해결하기 위해 캐시 상에서 적응형 패리티 그룹(APGOC : Adaptive Parity Group On Cache) 구성을 제안한다. 이 방법에서는 사용자 프로세스가 한 파일에 대한 접근을 커널에 요청할 때 파일 시스템의 파일 데이터 구조에 읽기/쓰기에 관련된 정보를 추가한다. 이 정보를 이용한 패리티 읽기를 통하여 데이터와 패리티 캐시를 연관 운영한다. 그리하여 캐시의 활용도를 높이고 디스크 요청에 대한 응답시간을 개선할 수 있다. 제안된 방법을 분석하고 시뮬레이션을 통하여 실험한 결과 기존의 방법에 비하여 6~13% 정도의 성능 향상을 관찰할 수 있었다.

Cache Management using a Adaptive Parity Group Configuration in RAID 5 Controller

Jung-Ho Huh[†] · Ja-Young Song[††] · Tae-Mu Chang[†††]

ABSTRACT

RAID 5 is a widely-used technique used to construct disk systems of high reliability and performance. This paper proposes APGOC (Adaptive Parity Group On Cache) organization on cache to solve "small write" problem of RAID 5 especially in OLTP (On-Line Transaction Processing System) environments. In our approach, when user process makes a request for a file to kernel, the information on the read/write characteristics is added to the file data structure of the file system. With this information, data and parity cache can be managed interchangeably through parity fetching. Therefore we can enhance the cache utilization and improve the disk request response time. Our method is analyzed and evaluated with a simulation method. Comparing with previous works, we observed about 6~13% of performance enhancement.

키워드 : 디스크 캐시(Disk cache), RAID 5 제어기(RAID 5 controller), 소규모 쓰기 문제(Small write problem), 캐시 적중률(Cache hit ratio)

1. 서 론

최근 프로세서의 처리속도는 매년 40~100%씩 증가되는데 비해 기계적인 부품을 가진 자기디스크의 성능 향상은 7% 정도에 불과하다. 따라서 중앙처리장치 및 주 기억장치와 디스크 입출력 속도의 차이를 극복하기 위한 방법으로, 자기 디스크의 신뢰도와 성능을 증가시키는 RAID(Redundant Arrays of Inexpensive Disks)가 널리 사용되어 왔다[1,2]. RAID는 신뢰도를 높이기 위한 패리티 등의 추가 정보의 배치 방법에 따라 여러 단계(level)로 나뉜다. 본 논문에서는 은행이나, 항공사, 우편 주문시 데이터 입력이나 거래조회 등과 같이 한번에 요청되는 데이터 크기가 작고 일정하며 입출력 횟수가 많은 트랜잭션 지향의 업무를 관리하는 OLTP(On-Line Transaction Processing) 환경에서의 RAID 5의 운영 방법을 다룬다. RAID 5의 패리티 연산은 높은 신뢰도와 디스크 공간 사용의 효율성 그리고 병렬 디스크 배열 형태로 인해 데이터 읽기에 높은 성능을 보이고, 데이터와 패리티를 분산 저장하여 신뢰도와 병렬성 면에서 OLTP 환경에 적합한 구조이다. 그러나 작은 단위의 쓰기 시에는 패리티 갱신을 위한 4~5번의 디스크 접근으로 인한 성능 저하로 쓰기 성능은 읽기 성능에 비해 떨어지는 단점이 있다[2, 3]. 이를 극복하기 위한 여러 가지 제안된 방법으로 패리티 로깅(parity logging), 부동 패리티(floating parity), 캐

† 준 회 원 : 동국대학교 대학원 컴퓨터공학과
†† 정 회 원 : 동국대학교 영상정보통신대학원 네트워크 관리학과
††† 종신회원 : 동국대학교 컴퓨터 · 멀티미디어공학과 교수
논문접수 : 2003년 1월 6일, 심사완료 : 2003년 5월 24일

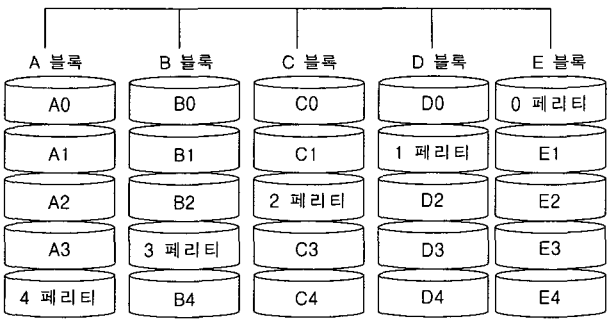
싱(caching)[4], 버퍼링(buffering)[5]등이 있는데, 이 중 가장 널리 이용되고 있는 것이 캐싱이다. 이는 일반적인 캐시에서 데이터와 패리티를 함께 캐시 내에 저장하나, 이들의 독립적인 운영으로 인해 쓰기시 패리티 연산을 위한 추가적인 패리티 접근이 필요하다.

본 논문에서는 RAID 5 제어기에서 사용자 프로세스가 한 파일에 대한 접근을 커널에 요청할 때, 파일 데이터 구조상에 그 파일에 대한 접근시 데이터와 패리티 캐시의 연관 운영에 필요한 읽기/쓰기에 관련된 정보를 추가하고, 이를 디스크 캐시의 운영에 이용함으로써 패리티를 읽는 캐시 운영 방법을 제안한다. 또한 쓰기 요청시에는 파일 시스템에서 쓰기에 관련된 정보를 이용하여 데이터와 패리티를 함께 캐시에 저장하여 추가적인 패리티 접근 문제를 완화시킨다. 기존의 방법과 본 논문에서 제안한 방법을 리눅스(Linux) 기반의 파일 시스템을 가정한 디스크 시뮬레이션을 통하여 성능을 비교하였다.

2. 관련 연구

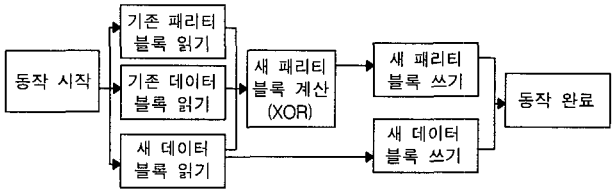
2.1 RAID 5의 구조 및 운영방법

RAID 5는 패리티 정보를 그룹 내 모든 디스크들에 분산 저장하여 패리티 디스크에 대한 병목현상을 개선하기 위한 구조이다. 접근될 데이터 블록과 패리티 블록이 서로 다른 디스크에 저장되어 있으므로 읽기나 쓰기 동작이 중첩될 경우 디스크 접근의 병렬성을 극대화 시킬 수 있다[3]. 이는 패리티 정보는 저장하지만 데이터를 중복 저장하지는 않으며 패리티 정보는 장애 발생시에 복구용으로 사용된다. 보통 3~5개의 디스크를 배열로 사용하며 다중 사용자 시스템에 적당하다[6]. RAID 5는 큰 크기의 읽기/쓰기가 디스크에 일어날 때는 효과적일 수 있으나 소규모 쓰기 작업일 때는 작업 부하에 비해서 오버헤드가 커지는 단점을 가진다. (그림 1)은 각 디스크에 패리티 블록이 좌 대칭 방식으로 배치된 RAID 5 형태이다[6].



(그림 1) RAID 5

RAID 5는 (그림 1)에서와 같이 패리티 블록을 좌 또는 우 대칭 방식으로 분산 저장하여 패리티 블록의 위치가 한쪽 끝까지 이동하였을 때 회전하는 특성을 가지며, 패리티 갱신 시 패리티 그룹이 기본 단위가 된다. 읽기시에는 한 패리티 그룹 내에서 요청한 블록들을 병렬적으로 동시에 읽어 실행되며 쓰기시에는 입출력에 대한 병렬성을 보장하는 읽기-수정-쓰기(read-modify-write)에 의해 동작한다. 즉, (그림 1)의 경우 A0, B0에 대한 쓰기를 할 때 (그림 2)와 같이 새로 기록될 블록 A0, B0와 기존에 기록되어 있는 블록 A0, B0 그리고 같은 패리티 그룹에 속하는 패리티를 XOR 연산하여 새로운 패리티를 생성한다[7].



(그림 2) 쓰기 동작

위와 같이 패리티 갱신시 4~5번의 디스크 접근으로 인해 OLTP 환경에서와 같은 소규모 쓰기의 경우 성능의 저하를 가져온다[7].

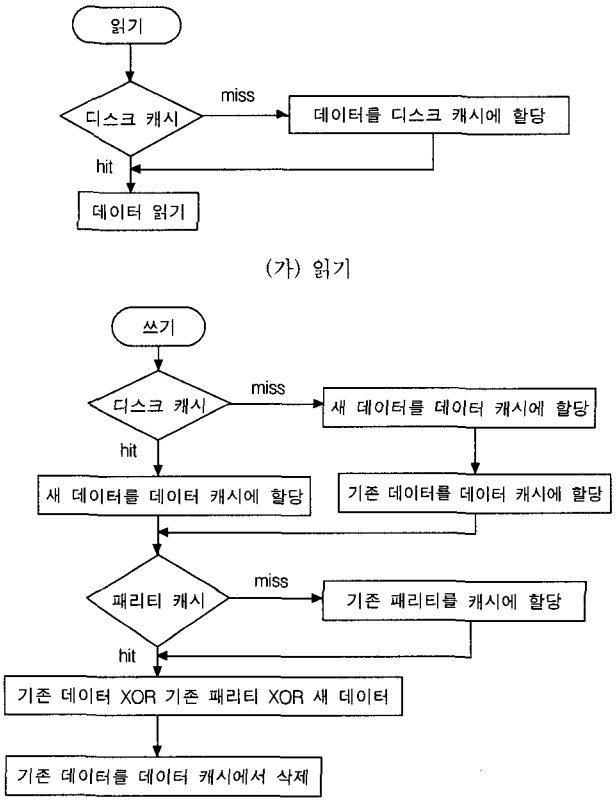
2.2 RAID 5의 성능향상을 위한 연구

읽기-수정-쓰기 방식을 사용하는 RAID 5의 쓰기 성능을 향상시키기 위한 여러 가지 방법들이 있다. 캐시를 이용해서 디스크 접근을 줄이고자 하는 방법[4]과 쓰기 방법을 바꿈으로 해서 디스크 기록시간을 줄이는 방법[7], 그리고 RAID 1에서 사용하는 디스크 반사 방식(disk mirroring)과 RAID 0에서 사용하는 스트리핑(striping) 방법을 혼합하여 사용하는 방법들이 있다. 디스크 수준의 쓰기 방법을 바꾸는 것으로는 부동 패리티, 패리티 로깅, 데이터 로깅(data logging), 핫 미러링(hot mirroring), 동적 스트리핑(dynamic stripping), AFRAID(A Frequently Redundant Array of Independent Disks)[7]등이 소개된 바 있으며, 캐시를 이용한 방법은 쓰기 캐싱(write caching)[4, 8], 쓰기 버퍼링(write buffering)[5], 다중 버퍼(multi buffer)[5], 패리티 캐싱(parity caching)[9] 등이 있다. 캐시는 현재에도 디스크 접근을 줄이는 방안으로 널리 쓰이며, 또 DRAM의 가격은 매 10년 만에 백분의 1로 떨어질 정도로 저렴해지는 추세이므로[10], 디스크 시스템에서 대용량의 캐시를 사용하는 경향이 높다. 따라서 본 논문에서도 대용량의 디스크 캐시를 가정하여 디스크 접근 시간을 줄이는 방안을 고려한다. 캐시를 이용한 방법에서 쓰기 캐싱과 쓰기 버퍼링은 디스크에 대한 여러 개의 작은 쓰기 동작을 버퍼에 모아서 하나의 큰 쓰기 동작으로 만들고 캐시를 이용하여 디스크의 접근을 줄이고자

하는 방법인데, 이는 데이터 갱신을 메모리에서 처리하고 디스크에 대한 효율적인 접근이 가능하지만 패리티 갱신이 고려되지 않는다는 단점을 가진다. 패리티 캐싱은 패리티 갱신만을 캐시로 처리하는 방법인데, 패리티 갱신 자체만의 부하는 감소시킬 수 있지만 데이터와 패리티의 연관성을 고려하지 않아 패리티에 대한 데이터 블록의 추가적인 작업부하가 발생할 수 있고, 다중 버퍼 역시 이와 유사한 문제를 갖고 있다.

2.3 기존의 RAID 5에서의 캐시 운영 방법

데이터 캐시와 패리티 캐시를 독립적으로 운영하는 것이 보통이며, 캐시 운영은 해싱 테이블을 이용하며 패리티 캐시는 패리티 블록을 데이터 캐시는 데이터 블록의 모임인 데이터 라인을 기본단위로 이용한다. 디스크에 대한 읽기/쓰기 요청 발생시의 캐시 동작은 (그림 3)과 같다.

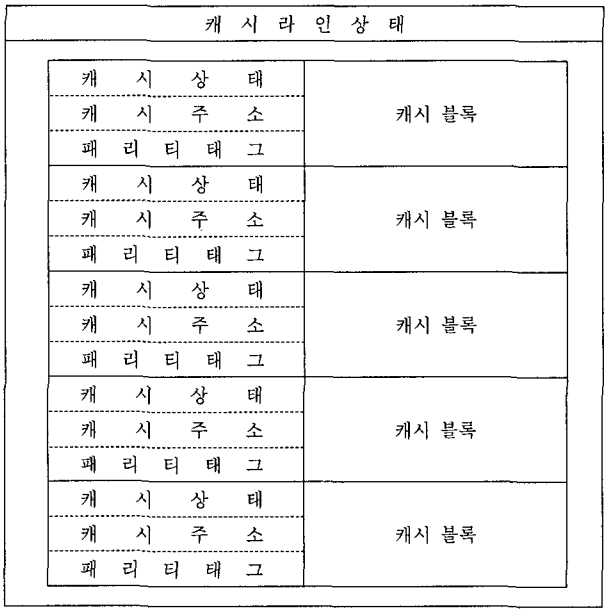


(그림 3) 디스크 데이터의 읽기 · 쓰기

디스크 쓰기 요청은 읽기-수정-쓰기 방식의 일반적인 RAID 5에서의 쓰기 방법에 따라 처리되며, 이 때 데이터 캐시의 데이터와 패리티 캐시의 패리티 사이의 독립운영으로 인해 패리티 연산을 위하여 추가적인 패리티 읽기가 필요하다. 그리고 디스크 접근 횟수를 줄이기 위해 블록 교체시 교체되는 블록을 디스크에 기록하는 지연쓰기(write back)방법이 이용된다. 패리티의 연산을 위하여 기존의 데이터 블록과 갱신될 새로운 데이터 블록 모두를 캐시 안에 저장 하고, 연산 후 기존의 데이터 블록은 캐시 안에서 삭제한다.

2.4 OLTP 환경에서 RAID 5의 캐시 운영

OLTP 환경에서 RAID 5의 효율적인 캐시운영 방법으로 (그림 4)와 같이 패리티 그룹 전체를 캐시에 적재하여 패리티 블록과 데이터 블록을 함께 연관시켜 캐시라인 단위로 운영하는 방법[9]이 있다. (그림 4)에서 캐시라인 상태는 적재되어 있는 패리티 그룹의 상태이며 캐시 상태는 각 캐시 블록의 상태이다.

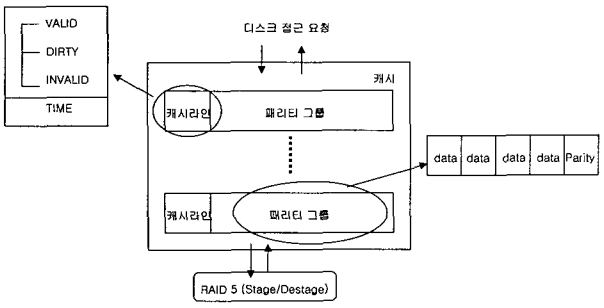


(그림 4) 캐시라인 구성

또한 캐시 주소는 캐시의 내용이 저장된 디스크의 주소이며 패리티 태그는 캐시 블록이 패리티인지를 나타내어 패리티 갱신시에 이 블록을 사용한다. 캐시의 운영은 캐시라인 상태와 라인 교체시 쓰기(Write on replacement)와 LRU 정책을 사용한다. 캐시라인 상태는 디스크로부터의 읽기/쓰기 여부, 캐시로의 쓰기 여부에 따라 변동되고 invalid 상태의 블록들이 교체의 최우선 순위가 된다. LRU 방식의 운영에 필요한 정보는 캐시라인에 기록되고 이 상태를 보고 관리한다. 캐시 운영에 필요한 Stage/Destage 동작은 모두 패리티 그룹으로 이루어진 캐시라인 단위로 이루어진다. (그림 5)는 이와 같은 동작을 나타내는 그림이다.

이 방법은 캐시라인 교체시에 패리티 그룹 단위로 읽기, 쓰기, 패리티 갱신 등을 모두 캐시 내에서 처리한다. 그러므로 디스크에 걸리는 작업 부하의 크기는 항상 일정하게 유지되고 캐시 내에서 블록과 패리티의 갱신이 이루어지므로 쓰기시에 패리티 갱신을 위해 추가적으로 디스크에 접

근하는 일반적인 캐시 운영 방법보다 효율적인 방법이다. 그러나 OLTP 환경과 같이 쓰기 작업보다 읽기 작업 비율이 높고 공간적인 지역성이 감소하는 경우에는 캐시 내에 있는 패리티 블록을 덜 사용하게 되어 캐시 용량의 낭비가 생긴다. 또한 블록의 크기를 나타내는 스트립의 크기가 적합하지 않을 경우에도 캐시 적중률이 떨어진다.

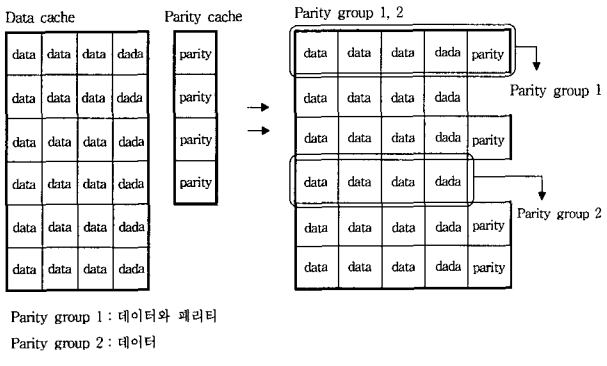


(그림 5) 캐시라인 운영

본 논문에서는 RAID 5의 단점으로 알려진 OLTP 환경에서와 같은 소규모 쓰기 문제를 해결하기 위한 방법으로, 이와 같이 패리티 그룹 전체를 캐시에 적재하여 패리티 블록과 데이터 블록을 연관시켜 함께 운영하는 방법과 기존의 RAID 5에서의 일반적인 캐시 운영방법을 혼합하여 사용한다. 그리하여 기존의 RAID 5에서 쓰기 시에 발생되는 추가적인 디스크 접근을 없애고 OLTP 환경에서와 같이 사용자 요청이 빈번하고 빠른 응답시간을 요구하는 경우에 좋은 결과를 보인다.

4. 캐시 상에서 적응형 패리티 그룹 구성(APGOC) 방법

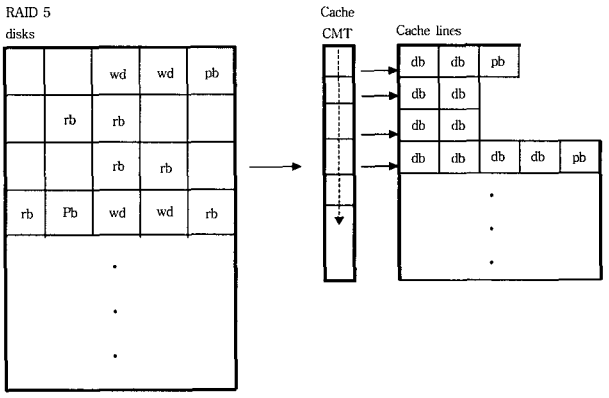
본 논문에서 제안한 리눅스 환경에서 적응형 패리티 그룹 구성을 통해 캐시 운영을 하는 방법인 APGOC를 설명한다. 일반적인 캐시 운영과는 달리 APGOC는 (그림 6)과 같이 데이터와 패리티를 가진 패리티 그룹 1과 데이터만을 가진 패리티 그룹 2로 운영한다.



(그림 6) 패리티 그룹 1, 2

이상적인 경우 APGOC에서 패리티 캐시의 크기는 스트립 크기 G를 가지는 RAID 5에서 1/G을 넘지 않으며 이것은 일반적인 캐시 운영 방법에서 적합하다고 알려진 패리티 캐시의 크기와 동일하다. 패리티 캐시는 데이터 캐시의 블록이 가지는 주소의 다음 주소를 가진다.

(그림 7)은 디스크에서 읽기 또는 쓰기 요청된 데이터와 패리티 블록들에 대해서 캐시관리 테이블(CMT : cache management table)을 이용하여 동적으로 캐시 메모리를 할당하여 논리적인 캐시라인을 구성하는 개념적인 구조이다.



(그림 7) APGOC의 논리적 캐시라인

데이터와 패리티를 한 개의 그룹으로 가지는 고정적인 패리티 그룹 방식은 읽기-수정-쓰기 방식의 캐시 운영 시에 읽기와 쓰기 두 개 이상의 패리티 그룹에 대한 동시 병렬성을 갖지 못한다. 넓은 스트립 크기를 가질 경우에도 불필요한 블록의 읽기로 인하여 동시에 읽을 수 있는 필요한 블록에 대한 읽기가 행해지지 못하는 경우가 발생하여, OLTP 환경에서와 같은 소규모 읽기/쓰기의 경우에 성능에 나쁜 영향을 줄 수도 있다.

4.1 논리적 캐시라인 유지

〈표 1〉 캐시 관리 테이블의 구성 요소

구성 요소	의 미
캐시상의 라인번호	논리적인 캐시라인의 순차 번호
그룹 태그	그룹 1, 2를 판단하는 1비트 태그
그룹 번호	실제 디스크에서 위치하는 패리티 그룹의 번호
전후방 포인터	이전 캐시라인의 리스트 시작 주소와 다음 캐시라인의 리스트 시작 주소
블록 위치 주소	패리티 그룹에 속하는 캐시 상에 존재하는 블록들 중 가장 우선하는 디스크 블록 번호를 가지는 블록의 캐시 블록 주소
패리티 블록 주소	패리티 캐시상의 패리티 블록의 주소, null이면 패리티 미존재
라인 상태 값	캐시 유지를 위한 라인의 상태 값

캐시 관리 테이블을 이용하여 동적으로 운영하는 캐시의 각 노드들은 캐시 상의 라인번호, 그룹 태그, 그룹 번호, 전후방 포인터, 블록 위치 주소, 패리티 블록 주소, 라인 상태 값 등이며 이들은 리스트 구조이다. 각각의 구성 요소에 대한 의미는 <표 1>과 같다.

그룹 1 읽기의 경우에 그룹 2 읽기로 읽어온 블록이 캐시 상에 존재하고 그것과 일치하는 패리티 블록이 패리티 캐시 상에 존재하지 않을 때에 패리티의 추가적인 읽기가 발생하는데 이를 크로스(cross) 읽기라고 정의하였다. 패리티 캐시의 경우 크로스 읽기를 위한 상태 비트를 가지며, 논리적 캐시라인을 이루는 각 블록의 구성요소는 태그, 전후방 포인터, 블록 상태, 캐시 블록 주소, 디스크 블록 주소 등이며 각각의 의미는 <표 2>와 같다.

〈표 2〉 블록 구성요소

구성 요소	의 미
태그	장치 번호와 관련 데이터 블록의 번호
전후방 포인터	이전 블록의 시작 주소와 다음 블록의 시작 주소
블록 상태	각 블록의 상태
캐시 블록 주소	캐시 상에서 블록이 위치하고 있는 장소의 주소
디스크 블록 주소	파일 시스템에서 사용하는 논리적인 주소

제어기 내에는 논리적 캐시라인의 존재여부를 위한 검색 및 유지, 즉 논리적 캐시라인의 삽입, 삭제, 연결과 블록의 삽입, 삭제, 연결 등을 행하는 프로세서와 RAID 5의 읽기-수정-쓰기 방식 및 캐시 관리 테이블에 필요한 메모리가 요구된다.

4.2 APGOC 캐시라인의 상태와 운영

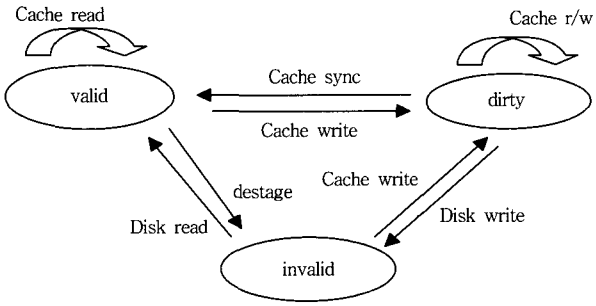
캐시라인은 다음의 세 가지 상태에 기반하여 운영된다.

① valid : 캐시 내에 존재하며 디스크와 내용이 일치하는 상태. 캐시 읽기(cache read) 동작시 상태를 유지하나, destage에 의해 invalid, 캐시 쓰기(cache write)에 의해 dirty 상태로 바뀐다.

② dirty : 캐시 내에 존재하나 캐시에서 수정되어 디스크와 일치하지 않는 상태. 캐시 읽기와 쓰기시 상태를 유지하나 동기화 동작에 의해 valid로, 디스크 쓰기(disk write) 동작에 의해 invalid로 변환된다. 동기화 동작은 디스크가 쉬고 있는 시간이 일정 수준 이상이거나 dirty 상태의 라인이 기준치 이상일 때 일어난다.

③ invalid : 캐시 내에 존재하지 않는 상태. 디스크 읽기에 의해 valid로, 디스크 쓰기에 의해 dirty로 바뀐다.

디스크 동작에 따른 상태 전이도는 (그림 8)과 같다.



(그림 8) APGOC의 캐시라인 상태 전이도

캐시라인의 교체는 LRU 정책이 기반이고 valid 상태에 있는 라인에 대하여 우선 적용한다. 이는 패리티 그룹 1, 그룹 2에 모두 동일하게 적용된다. 또 디스크에 대한 블록 갱신은 지연 쓰기를 이용한다. 크로스 읽기는 상태 전이도의 예외상황으로, 패리티 캐시 공간에 여분의 공간이 없을 경우 invalid 상태를 가지는 그룹 1 캐시라인의 그룹 태그를 그룹 2로 바꾸고, 여분의 패리티 블록에 읽어오는 새로운 패리티를 할당하여 해당 그룹 라인에 연결한 후 해당 라인의 태그는 그룹 1로 바꾸는 동작이 수반된다.

4.3 리눅스에서의 APGOC에 대한 지원

리눅스 운영체제의 커널은 사용자 프로세스의 파일 접근에 대한 요청이 들어오면 그룹 1, 그룹 2 캐시라인 운영을 위한 정보를 포함한 파일 데이터 구조[11]를 생성한다. 사용자 프로세스의 접근 권한과 inode에 존재하는 파일의 접근 권한을 비교하여 일치하는 inode의 권한에 쓰기 속성이 있을 경우는 그룹 1, 없을 경우는 그룹 2로 설정한다.

```
Algorithm initindex
Input : a. Working inode
        b. working file table
Output : none
{
 if (user id of current process == user id of inode)
    if (user permission of inode have W-permission)
        disk request index of file table = G 1 ;
    else
        disk request index of file table = G 2 ;
Else if (user group id of current process == group id of inode)
        if (group permission of inode have W-permission)
            disk request index of file table = G 1 ;
        else
            disk request index of file table = G 2 ;
Else
        if (other permission of inode have W-permission)
            disk request index of file table = G 1 ;
        else
            disk request index of file table = G 2 ;
}
```

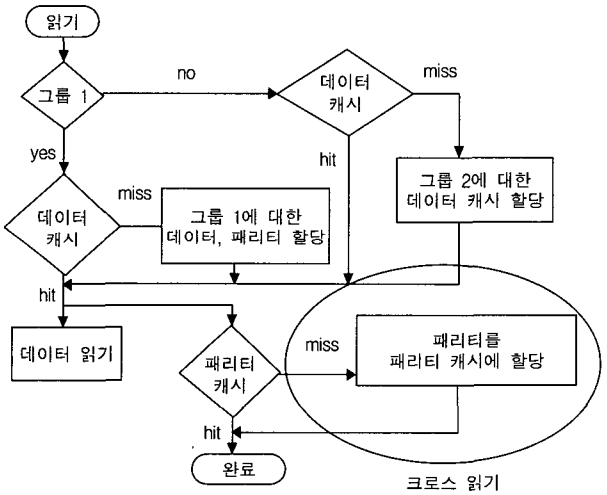
(그림 9) 그룹 1, 그룹 2 설정 알고리즘

(그림 9)의 알고리즘이 리눅스의 파일 열기 함수에 삽입되어 그룹 1, 그룹 2를 설정한다.

현재 프로세스의 그룹 id는 사용자 id의 그룹 id를 의미하며, 현재 프로세스의 사용자 id는 실제(real) id와 다른 파일을 접근하는데 필요한 유효(effective) id이다. 그 후 사용자가 접근하려는 블록의 그룹 1, 그룹 2에 대한 정보를 수정된 파일 데이터 구조인 파일 테이블에서 찾아 디스크 접근시에 함께 디스크 제어기에 주어진다. 읽기인 경우 읽기인 경우 호출되는 함수는 hd_out(dev, nsect, sec, head, cyl, cmd, &read_intr, grouptag)이고 쓰기인 경우 호출되는 함수는 hd_out(dev, nsect, sec, head, cyl, WIN_MULTWRITE, &multwrite_intr, grouptag) 또는 hd_out (dev, nsect, sec, head, cyl, WIN_WRITE, &write_intr, grouptag)이다. 수정은 그룹 1, 그룹 2에 대한 정보를 추가하여 줌으로써 이루어질 수 있다.

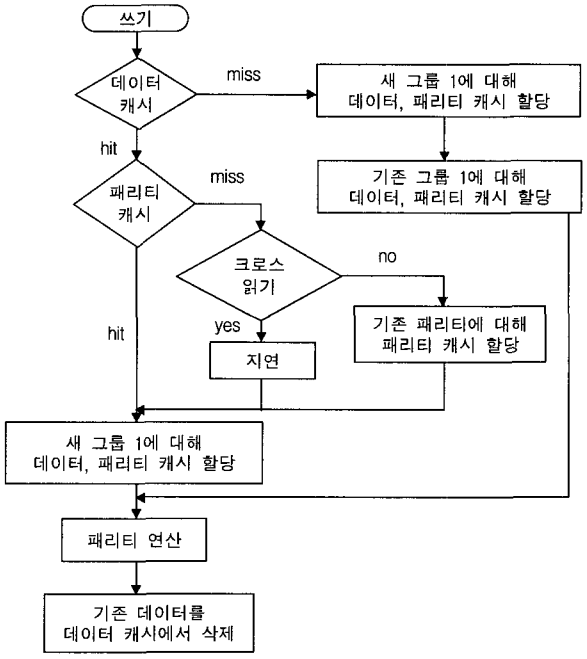
4.4 APGOC의 읽기, 쓰기

읽기의 경우 그룹 1 읽기, 그룹 2 읽기로 나뉘며 (그림 10)과 같다.



(그림 10) 디스크 데이터 읽기

그룹 1은 데이터 캐시에 데이터가 이미 존재하면 그 데이터를 읽고 존재하지 않으면 데이터와 패리티를 캐시에 할당하여 읽는다. 그룹 2도 그룹 1과 마찬가지지만 캐시에 데이터가 존재하지 않을 때 데이터만 할당한다는 점이 다르다. 크로스 읽기는 기존의 그룹화된 라인 단위 동작의 예외상황이며 이는 그룹 1 읽기의 경우에 그룹 2 읽기로 읽어 온 블록이 캐시 상에 존재하고 그것과 일치하는 패리티 블록이 패리티 캐시 상에 존재하지 않을 경우 발생한다. 이 때 패리티의 추가적인 읽기가 발생되는데 해당 캐시라인이 그룹 2에서 그룹 1로 바뀐다. 쓰기의 경우 (그림 11)과 같으며 일반적인 RAID 5의 쓰기 방법인 읽기-수정-쓰기 방식을 사용하여 동작한다. 읽기와 달리 쓰기는 캐시라인의 동작이 그룹 1로 고정되어 이루어진다.



(그림 11) 디스크 데이터 쓰기

5. 성능 분석

5.1 이론적 근거

본 논문에서 제안한 APGOC는 쓰기시에 발생하는 추가적인 패리티의 접근이 없다는 점이 장점이다. 특히 입출력 요청 빈도수가 많은 OLTP 환경에서의 쓰기로 인한 디스크 접근이 빈번한 환경에서 응답시간을 줄일 수 있다. 기존의 RAID 5에서 한 개의 데이터 블록의 읽기시간은 r, 한 개의 데이터 블록의 쓰기시간은 w, 전체 읽기시간은 R_{total}, 전체 쓰기시간은 W_{total}라 할 때 r, w, R_{total}, W_{total}은 다음과 같다[12].

$$r = 2R/D \tag{1}$$

$$w = 2R/D + (2R - 2R/D) + 2R/D \tag{2}$$

$$R_{total} = [(S+R)+r] + (m_2 - 1) * [(M+R)+r] + (m_1 - m_2 - 1) * [(H+R)+r] \tag{3}$$

$$W_{total} = 2 * [(S+R)+w] + 2 * (m_2 - 1) * [(M+R)+w] + 2 * (m_1 - m_2 - 1) * [(H+R)+w] \tag{4}$$

(단 D : 매 트랙의 데이터 단위들,
R : 평균 회전 지연(1/2 디스크 회전시간),
Bs : 접근할 데이터 수,
N : 디스크 배열 수,
m_1 : Bs/(N-1)D(헤드 스위치 시간),

m_2 : Bs/(N-1)DT(실린더 스위치 시간),
T : 매 실린더의 트랙,
S : 평균 탐색 시간,
H : 헤드 스위치 시간,
M : 한 트랙 탐색 시간)

기존의 RAID 5에서는 식 (2)에서와 같이 소규모 쓰기 시에 기존 데이터 읽기, 기존 패리티 읽기, 새 데이터 쓰기, 새 패리티 쓰기 등 총 2번의 쓰기와, 데이터가 이미 캐시에 있는 경우는 1번 읽기, 그렇지 않은 경우는 2번 읽기로 인해 전체적으로는 4번의 디스크 접근을 필요로 한다. APGOC에서는 쓰기 시에 기존 패리티의 추가적인 접근이 필요하지 않으므로 식 (2)에서 2R/D만큼의 시간이 줄어들며 데이터가 이미 캐시에 있는 경우에는 2R/D만큼의 시간이 더 줄어든다. 또한 식 (4)에서도 같은 효과를 보인다.

5.2 실 험

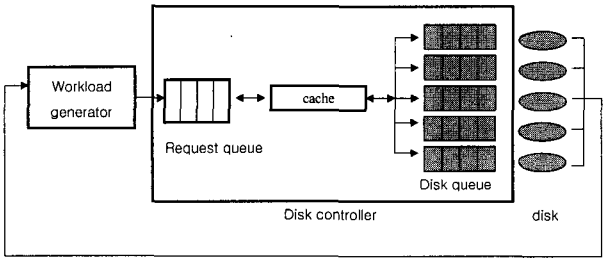
본 논문에서 사용한 시뮬레이터는 DiskSim 2.0[13]이며 본 논문의 특성에 맞도록 수정하여 사용하였다. DiskSim 2.0 시뮬레이터는 실행시간 면에서 효율적이고 다양한 입출력 장치 환경에서 사용할 수 있는 저장 장치 서브 시스템의 성능을 점검할 수 있는 시뮬레이터이다. 시뮬레이터는 크게 장치 구동기, 버스, 제어기, 디스크의 네 부분으로 되어 있으며, 스케줄러와 디스크 캐시, 제어기 상에 존재하는 요청 큐 등이 부가되어 있다. 작업 부하는 DiskSim 2.0의 내부 인공 트레이스 합성기능(synthetic trace)을 OLTP 환경[9, 14, 15]이 가지는 작고 빈번한 쓰기를 갖도록 수정하여 이용하였다. OLTP 환경의 한 모델인 항공 예매 시스템에서의 읽기/쓰기 측정치인 6 : 1로 가정하였고[16], 작업 요청 크기는 2KB로 하고, 캐시의 크기는 가격 대 성능비 측면에서 효율적이라고 볼 수 있는 0.2%로 하였다[9]. 또한 작업요청 간격 및 횟수의 변화에 따른 빈도수를 조절하여 평균 응답 시간을 통해 성능향상의 정도를 나타냈다. 이 작업 부하는 분석적 모형으로 계산한 값과 시뮬레이션 결과가 거의 동일한 것으로 입증된[17] 바 있다.

본 논문의 시뮬레이션은 디스크 요청 발생 부분과 캐시에 중점을 두고, 시뮬레이션에 필요한 그룹 1, 그룹 2 요청에 대해서는 파일 접근의 시간적 지역성을 고려하였다. 작업 부하 요청 생성기는 트레이스 데이터를 분석하여 쓰기 요청을 발견하면 그 쓰기 요청 시간으로부터 일정한 상한 하한 시간 사이를 검사하고, 쓰기 요청 블록 번호와 같은 블록 번호를 가지는 접근 요청이 검색되는 경우는 그룹 1로, 그 외의 경우는 그룹 2로 지정한다. 작업 부하의 특성과 시뮬레이션 변수는 <표 3>과 같으며 시뮬레이션한 디스크의 주요 특성을 함께 나타냈다.

〈표 3〉 작업부하의 특성과 시뮬레이션 변수

workload parameter request interval 평균 : 24.24ms 표준편차 : 27.46ms 최대간격 : 444.8ms request size Fixed 2KB read write ratio 6 : 1 request number 매 시뮬레이션마다 200,000회
cache parameter strip unit size 2KB cache size 0.2% scheduling algorithm LRU
array parameter head scheduling FCFS disk number 5
disk parameter disk name HPC 3323A cache max size 512B blocks disk scheduling LBN based C-LOOK[13]
disk(HPC3323A)의 주요 특성 max capacity : 1.03GB geometry : 2982 cyls surface 7 sector size 512B 8zone seek time : 0.42 · (dist - 1)^(1/2) + 0.01 · (dist-1) + 1.178(ms) average seek time : 10ms, rotation speed : 5400rpm

(그림 12)는 시뮬레이터 모듈이고, <표 4>는 입력 트레이스에 따라 평균 접근 요청 간격에 변화를 주어 실험한 입력 데이터 현황이다.



(그림 12) 시뮬레이터 모듈 구성

〈표 4〉 시뮬레이션 입력 데이터 현황

트레이스	평균 접근 요청 간격(ms)	초당 접근 요청회수(회)
A	37.97	26.34
B	23.27	42.97
C	16.16	61.88
D	10.51	95.15
E	6.60	151.52
F	3.30	303.03

<표 4>에서 초당 접근 회수는 평균 접근 요청 간격의 역수이다. OLTP 환경에 따라 단위 시간당 입출력 요청의 수는 다양할 수 있으므로 본 논문에서는 다양한 요청 간격을 시뮬레이션 매개 변수로 사용하였다. 사용된 트레이스의

평균치를 구하여 <표 4>에서 나타냈고, 읽기/쓰기 비율은 OLTP 환경에 맞게 일관되게 6 : 1로 동일하게 적용하였다. <표 5>는 기존의 방법과 본 논문에서 제안된 APGOC의 적중률을 비교하기 위한 것으로 캐시 적중률은 캐시의 효율성을 직접적으로 반영하여 측정할 수 있는 시스템 성능을 나타내는 중요한 요소가 된다[18].

〈표 5〉 캐시 적중률

트레이스	적중률	
	기존	APGOC
A	0.75	0.89
B	0.77	0.94
C	0.79	0.88
D	0.77	0.94
E	0.78	0.90
F	0.76	0.93
평균적중률	**0.77**	**0.91**

〈표 6〉 처리 시간

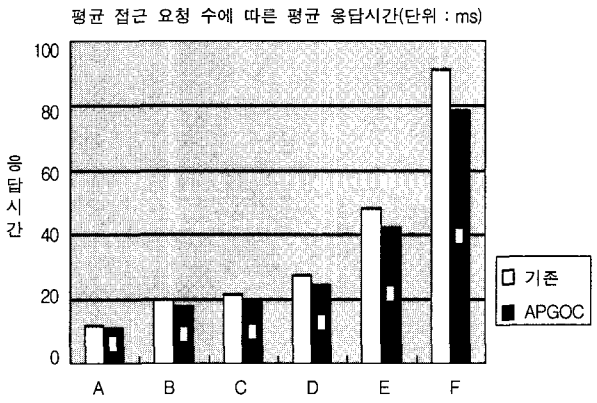
Param eters	Time(ms)
Read hit after read	0.35
Read hit after write	1.39
Read miss after read	2.50
Read miss after write	2.50
Read after read	**3.29**
Read after write	2.92

<표 5>는 각 트레이스에 따른 캐시 적중률의 평균치로 본 논문에서 제안된 APGOC의 전체적인 평균 적중률이 0.91로 기존의 방법인 0.77에 비해서 각 트레이스에 일관되게 향상된 것을 확인할 수 있다. 적중률이 높아짐에 따라 캐시의 활용도가 높아지므로 디스크에 대한 추가적인 접근이 줄고 따라서 응용 프로그램의 요청에 대한 응답시간이 감소하게 된다. <표 6>은 각 트레이스에서 발생된 평균적인 처리 시간을 나타낸 것이다. 전형적인 OLTP 환경의 응용인 ERP-DB[15]의 경우 Write after read의 비율이 14% 미만임을 고려하여 균등 분포하여 실험하였으며 <표 6>의 처리시간은 <표 7>의 평균 응답시간에 포함된 시간이다. <표 7>은 응답시간 감소율을 강조하기 위하여 수치로 나타낸 표이며, 크로스 읽기의 발생율에 따른 평균응답시간 변화율도 함께 나타냈다. 그러므로 각 트레이스에 대하여 평균 응답 시간 면에서 APGOC와 기존 캐시 운영 방법을 비교할 때 APGOC가 A 트레이스는 6.34%, F 트레이스는 13.25%로 약 6~13%가량 성능이 향상됨을 알 수 있다. 또한 크로스 읽기로 인한 추가적인 시간지연은 전체적인 평균 응답 시간에 큰 영향을 주지는 못 하였음을 확인하였다.

〈표 7〉 응답시간 감소율

평균 응답 시간(ms)				크로스 읽기	
트레이스	기존	APGOC	**감소율**	발생율	평균응답시간 변화율
A	11.98	11.22	**6.34%**	0.11%	0.01%
B	19.65	18.03	**8.23%**	0.46%	0.12%
C	21.31	19.06	**10.56%**	0.78%	0.48%
D	27.65	24.53	**11.27%**	1.67%	1.33%
E	48.02	41.98	**12.58%**	2.33%	2.07%
F	90.75	78.73	**13.25%**	4.95%	2.34%

(그림 13)은 <표 5>의 평균 응답시간을 그림으로 나타낸 것이다. 응답시간은 장치 구동기에서 출발한 요청이 다시 장치 구동기에서 완료될 때까지이며, 제어기 큐에 있는 시간, 버스 전송 시간, 디스크 I/O 시간을 모두 포함한 값이다. 기존의 캐시 운영 방법에 비해 본 논문에서 제안한 APGOC를 사용하는 경우 응답시간 면에서 개선됨을 확인할 수 있다. 특히 사용자 요청이 빈번하고 빠른 응답시간이 요구되는 경우 개선의 정도가 더 커진다는 것을 알 수 있다.



(그림 13) 응답시간 측정 결과

본 논문에서 제안한 APGOC는 파일 시스템에 읽기/쓰기에 대한 정보를 추가하여 기존의 RAID 5에서의 방법에 비해 쓰기시에 기존 패리티를 패리티 캐시에 할당하는 추가적인 패리티 접근이 발생하지 않게 되어 응답시간을 줄이고자 한다. 또한 패리티 그룹 전체를 캐시에 적재하는 RAID 5의 캐시 운영 방법에 비해 읽기 작업 비율이 높은 OLTP 환경에서도 좋은 결과를 보인다. 이는 파일 시스템의 파일 데이터 구조에 추가한 읽기/쓰기 정보로 패리티 읽기를 하여 데이터와 패리티 캐시를 연관 운영한 결과이며, 디스크 접근 시간의 감소와 캐시의 활용도를 높여 디스크 요청에 대한 응답시간을 개선하고자 한다.

6. 결 론

본 논문은 적응형 패리티 그룹 구성을 이용한 RAID 5의 캐시 운영 방법을 제안하였다. 이로써 OLTP 환경과 같은 소규모 쓰기시 추가적인 패리티 읽기가 일어나지 않게 되어 시간지연의 문제 해결은 물론 캐시의 활용 또한 높일 수 있다. 그리하여 작은 크기의 빈번한 읽기/쓰기시에 보다 효율적인 성능을 보임을 알 수 있다. DiskSim 2.0을 수정하여 이용한 시뮬레이션 결과, 본 논문에서 제시한 캐시상의 적응형 패리티 그룹 구성 방법이 기존의 RAID 5 이용 방법에 비해 6~13% 정도의 평균적인 응답시간을 줄일 수 있었다. 앞으로의 작업으로는 크로스 읽기로 인한 추가적인 시간지연의 문제와 본 논문에서 제시한 방법을 수학적인 이론으로 구체화시켜 분석적 모형을 만들고 여러 가지 다양한 작업 부하를 사용한 시뮬레이션을 통해 비교 연구해 볼 수 있다. 또한 본 논문에서 제시한 방법을 OLTP 환경 이외의 다양한 작업환경에 적용하여 보다 효율적인 캐시 운영 방안을 연구할 수 있다.

참 고 문 헌

[1] S. Chen and D. Towsley, "A Performance Evaluation of RAID Architectures," IEEE Transactions on Computers, Vol.4, No.10, pp.1116-1129, 1996.

[2] A. K. Sahai, " Performance Aspects of RAID Architectures," IEEE International Conference on Performance Computing and Communications, pp.321-327, 1997.

[3] R. Recio and W. T. Boyd, "Methodology to Optimize the Cost/performance of Disk Subsystems," IEEE International Symposium on Performance Analysis of Systems and Software, 2000.

[4] J. H. Kim, S. W. Eom, S. H. Noh and Y. H. Won, "Striping and buffer Caching for Software RAID File Systems in Workstation Clusters," Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, pp.544-551, 1999.

[5] K. A. Hua, K. Vu and T. H. Hu, "Improving RAID Performance Using a Multibuffer Technique," Proceedings of the 15th International Conference on Data Engineering, pp.79-86, 1999.

[6] S. C. Chau and A. W. C. Fu, "A Gracefully Degradable Declustered RAID Architecture with Near Optimal Maximal Read and Write Parallelism," Proceedings of the IEEE International Conference on Cluster Computing, pp.309-318, 2000.

[7] E. Gabber and H. F. Korth, "Data Logging : A Method for Efficient Data Updates in Constantly Active RAIDs," Proceedings of the 14th International Conference on Data Engineering, pp.144-153, 1998.

[8] B. Y. Kim and Y. S. Chang, "Improved RAID5 Controller Using Load-Balanced Destage Algorithm," The 5th Electronics Letters, Vol.34, No.3, Feb., 1998.

[9] 이정민, 장태무, "OLTP 환경에서 RAID 레벨 5의 효율적인 캐시 운영 방법에 관한 연구", 한국정보과학회 춘계학술발표 논문집, 제23권 제1호, pp.351-354, 1996.

[10] J. Gray and R. Shenoy, "Rules of Thumb in Data Engineering," Technical Report MS-TR-99-100, MicroSoft Research, 2000.

[11] M. Beck, H Bohme, M. Dziadzka, U. Kunitz, R. Magnux and D. Verworner, "Linux Kernel Internals," 2nd edition, Addison-Wesley, 1998.

[12] H. Jin and K. Hwang, "Stripped mirroring RAID architecture," Journal of Systems Architecture 46, pp.543-550, 2000.

[13] G. R. Ganger, B. L. Worthington and Y. N. Patt, "The DiskSim Simulation Environment Version 1.0 Reference Manual," Technical Report CSE-TR-358-98, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1998.

[14] S. K. Mishra and P. Mohapatra, "Performance Study of RAID-5 Disk Arrays with Data and Parity Cache," International Conference on Parallel Processing, pp.222-209, 1996.

[15] W. Hsu, et.al, "Characteristics of I/O Traffics in Personal Computer and Server Workloads," UCB/CSD-02-1179, EECS Univ. California, Berkeley, April, 2002.

[16] J. Menon and D. Mattson, "Performance of Disk Arrays in Transaction Processing Environments," International Conference on Distributed Computing Systems, pp.302-309, June, 1992.

[17] G. R. Ganger, "System-Oriented Evaluation of I/O Subsystem Performance," Technical Report CSE-TR-243-95, Department of EECS, University of Michigan, Ann Harbor, June, 1995.

[18] C. Yun, Y. Genke, W. Zhiming, "The Application of Two-Level Cache in RAID System," Proceedings of the 4th world Congress on Intelligent Control and Automation, June, 2002.

허 정 호

e-mail : hjh99@dgu.ac.kr

1984년 동국대학교 학사(전산)

1987년 한양대학교 석사(전기 · 전자공학)

2001년 동국대학교 박사수료(컴퓨터 공학)

관심분야 : 입출력 시스템, 컴퓨터 구조, 병렬처리

송 자 영

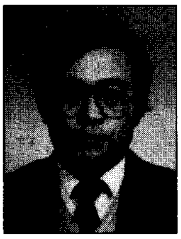
e-mail : sosong@nownuri.net

1998년 호서대학교 학사(전산)

2000년 동국대학교 석사(컴퓨터 공학)

2003년 동국대학교 영상정보통신 대학원 박사과정(네트워크 관리학)

관심분야 : 분산운영체제, 운영체제보안, IPV6



장 태 무

e-mail : jtm@dgu.ac.kr

1977년 서울대학교 전자공학과(학사)

1979년 한국 과학기술원 전산학과(이학석사)

1995년 서울대학교 컴퓨터 공학과(공학박사)

1979년~1981년 한국 전자 기술 연구소 연구원

1998년 University of Southeastern Lousiana 교환교수

1981년~현재 동국대학교 컴퓨터·멀티미디어 공학과 교수

관심분야 : 분산 및 병렬 처리, 컴퓨터 구조, 입출력 시스템