파일 접근 패턴과 캐쉬 영역을 고려한 선반입 기법

임 재 덕[†] · 황보 준 형[††] · 고 광 식[†††] · 서 대 화[††††]

요 약

디스크 I/O성능 개선을 위한 여러 캐싱과 선반입 알고리즘이 연구되어져 왔다. 선반입 알고리즘은 디스크 접근 횟수를 줄임으로써 병렬파일시스템의 I/O성능을 높여준다. 본 논문에서는 OBA 선반입 기법의 확장 버전인 AMBA 선반입 기법을 제안한다. AMBA 선반입 기법은 현재 사용되고 있는 다음 블록 하나만 선반입하는 것이 아니라, 디스크의 대역폭이 충분하다면 여러 파일 블록을 연속해서 선반입 하는 방법이다. 이는 응용프로그램에서의 데이터 요청이 빈번해도 이전에 연속해서 선반입한 파일 블록에 의해 선반입 효과를 기대할 수 있다. 그리고 버퍼캐쉬 영역 내에 선반입된 파일 블록의 수를 제한함으로써 버퍼캐쉬의 효율성을 높여준다. 제안된 선반입 기법은 리눅스 운영체제 기반의 사용자 수준의 파일 시스템에서 구현하고 실험하였다. 실험 결과 제안된 AMBA 선반입 기법은 순차적으로 접근되는 큰 파일에 대해서는 기존 시스템에 비해 30~40%의 성능을 개선해준다.

Prefetching Policy based on File Acess Pattern and Cache Area

Jae-Deok Lim[†] · Jun-Hyoung Hwang-Bo[††] · Kwang-Sik Koh[†††] · Dae-Wha Seo[††††]

ABSTRACT

Various caching and prefetching algorithms have been investigated to identify an effective method for improving the performance of I/O devices. A prefetching algorithm decreases the processing time of a system by reducing the number of disk accesses when an I/O is needed. This paper proposes an AMBA prefetching method that is an extended version of the OBA prefetching method. The AMBA prefetching method will prefetch blocks continuously as long as disk bandwidth is enough. In this method, though there were excessive data request rate, we would expect efficient prefetching. And in the AMBA prefetching method, to prevent the cache pollution, it limits the number of data blocks to be prefetched within the cache area. It can be implemented in a user-level File System based on a Linux Operating System. In particular, the proposed prefetching policy improves the system performance by about 30~40% for large files that are accessed sequentially.

키워드 : 파일선반입(file prefetching), 캐싱(caching), 병렬파일 시스템(parallel file system), 운영체제(operating system)

1. 서 론

최근 컴퓨터 시스템의 발전 경향을 보면 프로세서 등과 같은 컴퓨팅 장치의 성능은 비약적인 발전을 이룬데 반해 저장장치의 입출력 성능은 괄목할 만한 발전을 이루지 못했다. 사용되는 데이터들이 대부분 디스크를 통해 작업이 이루어지므로 컴퓨팅 능력이 아무리 뛰어나다 하더라도 저장장치의 입출력 처리 성능이 따라주지 않는다면 전체 시스템의 성능은 떨어질 수밖에 없다.

이런 입출력 처리 문제를 해결하기 위해 RAIDs(Redundant Array of Inexpensive Disks)와 같은 하드웨어적인 방법들을 이용하지만, 입출력 포트의 대역폭 제한으로 입출력 성능 향상에 한계가 있다. 따라서 병렬 파일 시스템이나 클러스터 파일 시스템과 같은 소프트웨어적인 해결책이 요구되고 있다. 그리고 이러한 특수 파일 시스템의 성능을 더욱 높이기 위해서 파일 캐싱이나 파일 선반입 기법들이 연구되어지고 있다.

파일 접근의 지역성이 클 경우, 디스크 버퍼캐쉬의 사용은 파일 블록의 입출력 횟수를 줄일 수 있어서 실질적인 파일 접근 시간을 줄일 수 있다. 하지만 동영상이나 멀티미디어 자료의 경우, 파일의 크기가 매우 크고, 파일 블록들이 거의 한 번만 읽혀지면 버퍼캐쉬의 효율성 문제가 발생한다. 이런 한계를 보완하여 파일 접근 시간을 줄이기 위한 방법으로, 파일 블록의 선반입 기법을 사용한다[3-5]. 특히 대용량의 VOD, AOD 그리고 MOD 같은 멀티미디어 데이터의 서비스에서 데이터 파일을 액세스하는 패턴이 순차적이면서 한번 읽혀진 파일 블록들은 대부분 다시 읽혀지지 않는다는 특성이 있다. 이런 특성을 가지는 파일 액세스 형태에서는 버퍼캐쉬의 크

† 정 회 원 : 한국전자통신연구원 연구원
†† 준 회 원 : 경북대학교 대학원 전자공학과
††† 정 회 원 : 경북대학교 전자공학과 교수
†††† 종신회원 : 경북대학교 전자전기공학부 교수

논문접수 : 2001년 3월 20일, 심사완료 : 2001년 9월 27일

기가 파일의 크기만큼 크지 않은 이상 버퍼캐쉬의 효율성을 기대할 수 없으며, 파일 선반입 기법이 시스템 성능 향상에 많은 기여할 것으로 본다.

하지만 무조건적인 파일 블록의 선반입은 버퍼캐쉬의 효율성을 저하시킬 수도 있다. 파일 접근 형태가 순차적이지 않을 경우, 선반입된 파일 블록들이 사용될 확률은 적어지고, 버퍼캐쉬 내에 이미 존재하고 있으며 곧 사용될 파일 블록들을 교체해 버릴 가능성이 커진다. 따라서 버퍼캐쉬 내에 존재하는 파일 블록과 선반입된 파일 블록들이 각각 관리되기보다는 파일 접근 형태(file access patterns)과 버퍼캐쉬 영역을 고려하여 선반입되는 파일 블록들의 양을 조절할 수 있는 통합적인 관리가 필요하다[6, 9]

본 논문에서는 병렬 파일 시스템(PFSL)의 파일 입출력 처리의 효율성을 극대화하기 위해서, 버퍼캐쉬 영역에 선반입될 수 있는 파일 블록들의 최대 개수를 동적으로 제한하고, 파일 블록들의 개수를 동적으로 변화시킬 수 있는 AMBA (adaptive multiple-block ahead) 선반입 기법을 제안한다[8]. AMBA 선반입 기법은 OBA(One-Block Ahead) 선반입 기법을 확장한 것으로, 대형 과학 계산 응용에서 사용되는 파일이나 VOD와 같은 멀티미디어 응용에 사용되는 파일의 접근 형태의 90% 이상이 순차적인 접근이라는 지금까지의 연구 논문 결과를 바탕으로 하고 있다[1, 7].

본 논문에서 제안한 AMBA 선반입 기법의 성능을 확인하기 위해서 리눅스 클러스터 환경에서 구축된 병렬 파일 시스템(PFSL)에 직접 구현하고 실험하였다. 파일에 대한 접근 형태가 순차적일 경우 선반입 기법을 적용하였을 때 병렬 파일 시스템의 입출력 처리 성능이 크게 향상되는 것을 확인하였다.

본 논문은 서론에 이어 2장에서 기존 선반입 기법에 대한 문제를 제기하고, 3장에서 제안한 선반입 기법을 설명한다. 4장에서는 제안된 선반입 기법의 성능을 기존의 선반입 기법의 성능과 비교 및 분석하였다. 마지막으로 5장에서는 실험 결과를 바탕으로 결론을 맺는다.

2. 파일 블록 선반입 기법

기존에 사용된 선반입 기법의 대표적인 것들은 다음과 같다. 선반입 알고리즘 중 가장 간단하면서 가장 많이 사용되는 기법으로 OBA(One Block Ahead) 선반입 기법이 있다[5]. OBA 선반입 기법은 요청된 파일 블록 다음의 파일 블록을 선반입하는 방법으로써 파일의 액세스 패턴 형태가 순차적일 경우 시스템의 성능 향상에 큰 역할을 한다.

파일의 액세스 형태는 시스템보다 그 파일을 사용하는 응용프로그램이 더 잘 알고 있으므로, 응용프로그램이 사용할 파일의 액세스 형태의 정보를 시스템에 알려줄 수 있는 좋은 구조가 제공된다면 좀 더 효율적인 선반입이 이루어질 수 있다. 이런 원리를 기반으로 Scotch file system에서 TIP(Transparent Informed Prefetching) 기법이 제안되었다[3].

데이터 압축에 사용되는 알고리즘들은 예측 구조로써 잘 알려져 있으며, 몇몇 선반입 기법들이 데이터 압축 알고리즘을 기반으로 하여 제안되었다[4]. 이들 모두는 확률 트리 및 확률 그래프 형태의 예측 구조를 구성한다. 이 예측 구조는 현재 요청된 파일 블록 다음의 블록들의 사용 확률에 대한 정보를 가지고 있어 시스템이 다음에 사용될 파일 블록을 예측할 수 있도록 해준다.

ISG(Interval-and-Size Graph) 선반입 기법은 데이터 압축을 기반으로 하는 선반입 기법을 간소화한 것으로써, 응용프로그램은 대부분 정규적인 형태로 파일을 액세스한다는 이론을 바탕으로 한 알고리즘이다[5]. 이 기법에서는 선반입에 필요한 정보를 그래프 형태로 유지하고 있다. 따라서 이 그래프가 형성된 파일에 대한 액세스가 이루어 질 경우 이 그래프를 이용하여 다음에 사용될 파일 블록을 요청할 수가 있게 된다.

위에서 설명된 여러 가지 알고리즘들은 앞으로 사용될 블록들을 예측하여 미리 가져옴으로써 디스크 입출력의 시간을 줄이는데 효과적인 성능을 나타낸다. 하지만 대부분의 알고리즘들이 버퍼캐쉬 히트 정도를 높이는데 집중되어 있다.

또한 선반입된 파일 블록이 버퍼캐쉬 내에 존재하고 있던 기존의 파일 블록들을 어떻게 교체하는지, 또 선반입된 파일 블록들이 캐싱될 때 버퍼캐쉬 내에 존재하고 있던 기존의 파일 블록들을 어느 정도 유지시켜주는지에 대한 관리 즉, 선반입된 파일 블록들과 캐싱된 파일 블록들간의 통합적인 관리가 이루어져 있지 않다. 그리고 다음에 사용할 때 사용할 블록을 예측하기 위한 복잡한 자료 구조 등을 유지하고 매번 갱신해야 하며, 사용할 파일의 액세스 패턴을 얻어내기 위해 최소한 한 번은 파일에 대한 액세스가 이루어져야 하는 단점이 있다.

최근 병렬 및 분산 시스템의 응용프로그램에서 사용되는 파일의 액세스 패턴에 대한 연구 결과를 보면 대부분이 순차적이며 그 크기도 대용량화되고 있다[1, 2, 7]. 따라서 굳이 복잡한 알고리즘으로 다음 사용될 파일 블록을 예측한다는 것은 오히려 시스템에 과부하로 작용할 가능성이 있고, 현재 사용된 파일 블록을 기반으로 다음에 사용될 파일 블록을 읽어 온다면 미리 사용될 파일의 액세스 패턴을 유지할 필요도 없어진다.

따라서 디스크 대역폭이 여유가 있다면 미리 설정해둔 전체 버퍼캐쉬 영역 한도 내에서 계속적인 선반입을 한다. 이는 차후의 응용프로그램 요청 사이의 시간 간격이 짧아도 버퍼 캐쉬 영역 안의 선반입된 다수의 파일 블록으로 인해 디스크 대역폭이 충분하지 않을 때 선반입을 하지 않아도 선반입 효과를 얻을 수 있다.

다음 장에는 이러한 기능이 포함된 AMBA 선반입 기법에

대해서 기술한다.

이런 이유로 본 논문에서는 시스템의 선반입 기법으로 OBA 기법의 문제점을 개선하고, 기능을 확장한 AMBA(Adaptive Multiple Block Ahead) 기법을 제안하고 병렬 파일 시스템에 적용하였다.

3. AMBA 선반입 기법

3.1 OBA 선반입 기법

OBA 선반입 기법에서는 응용프로그램이 사용할 다음 파일 블록은 현재 사용된 파일 블록 다음블록일 것으로 가정한다. 따라서 대용량 연속데이터를 처리하는 병렬 및 분산 응용프로그램에서 OBA 선반입 기법은 매우 효율적인 파일 선반입 방법이다. 이는 병렬 및 분산 응용프로그램에서 사용되는 파일들의 평균적인 크기가 일반 시스템에서 사용되는 파일들의 크기보다 크며, 응용프로그램에서 파일을 사용할 때 액세스 패턴이 순차적인 경우가 많기 때문이다.

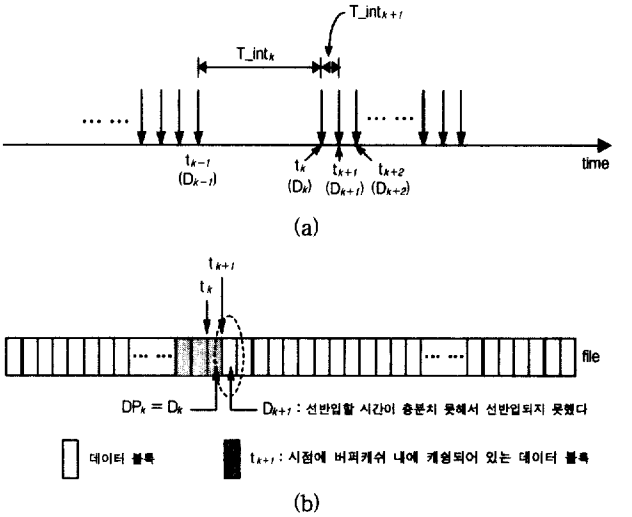
하지만 OBA 선반입 기법에서는 응용프로그램이 파일 블록에 대한 읽기 요청을 할 경우에만 다음의 파일 블록을 선반입 한다. 즉, 응용프로그램에서 요청한 파일의 파일 블록을 시스템이 서비스해 줄 때, 다음의 파일 블록을 선반입 해주는 것이다. 이는 디스크 대역폭과 무관하게, 응용프로그램이 새로운 파일 블록을 요청하면 항상 선반입이 수행된다는 의미이다. 그리고 응용프로그램에서 파일 블록을 요청하는 시간적 간격을 두지 않고 연속적으로 파일 블록을 요청할 경우, OBA 기법은 선반입 효과가 떨어진다. (그림 1)은 OBA 선반입 기법의 문제점을 도식화한 것이다.

다음은 사용될 용어를 정의한다.

- D_i : 파일 내의 i 번째 파일 블록
- t_i : 응용프로그램에서 D_i를 요청하는 시간
- T_int_i : 응용프로그램에서 D_{i-1}을 요청한 후 D_i를 요청하기까지의 시간적 간격($T\_int_i = t_i - t_{i-1}$)

(그림 1)(a)에서 처럼 T_int_i가 충분히 길 경우 즉, 각각의 파일 블록들의 요청이 시간 간격을 가지면서 요청될 경우이다. 응용프로그램은 시각 t_k에 D_k를 사용하기 위해 D_k를 요청하면, D_k는 이미 버퍼캐쉬에 선반입되어 있기 때문에 버퍼캐쉬에서 데이터를 불러서 사용할 수 있다. 그리고 다음 T_int_{k+1} 후인 시각 t_{k+1}에 D_{k+1}을 사용하기 위해 D_{k+1}을 선반입 요청한다. (D_k)를 사용하는 동안 D_{k+1}이 선반입될 시간(T_int_k)이 충분하므로, 응용프로그램에서 다음 파일 블록(D_{k+1})을 요청하기 전에 그 파일 블록(D_{k+1})이 선반입되어 있으므로 응용프로그램에서는 디스크의 입출력없이 파일 블록을 사용할 수가 있다.

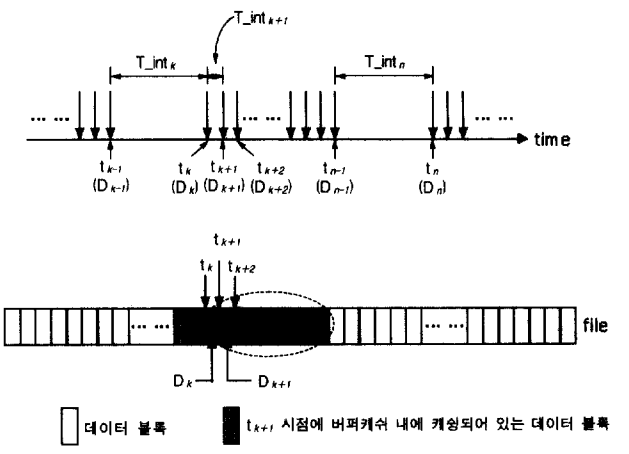
하지만 (그림 1)(b)처럼 응용프로그램에서 D_{k+1}이 선반입되어 버퍼캐쉬에 저장되기 전에 D_{k+1}을 요청하게 되면, 즉 요청간격 T_int_i가 충분히 길지 않으면 선반입 효과가 낮아지게 된다. t_{k+1} 시간에는 사용할 파일 블록(D_{k+1})이 선반입되어 있지 않아 시스템은 디스크 입출력을 통해 파일 블록을 응용프로그램에게 서비스해 주어야 한다. 이럴 경우가 많이 발생한다면 선반입에 의한 효과를 볼 수가 없다. 따라서 선반입할 시간이 충분히 주어졌을 경우 되도록 많은 파일 블록을 선반입함으로써 선반입 효과를 증대시킬 수 있다.



(그림 1) OBA(One Block Ahead) 선반입 기법의 문제점

3.2 AMBA 선반입 기법

선반입 효과를 증대시키기 위해 OBA 선반입 기법을 확장한 AMBA(Extended-OBA) 선반입 기법을 제안한다. AMBA 선반입 기법은 응용프로그램의 요청과 관계없이 디스크 대역폭에 여유가 있다면 정해진 파일 블록의 개수만큼 계속적인 선반입을 수행하는 기법이다. 선반입할 파일 블록의 개수를 제한하는 것은 무분별하게 파일 블록들이 선반입되어 버퍼캐쉬 내에 있는 기존의 선반입된 파일 블록들이 교체되는 것을 막기 위한 것이다.



(그림 2) AMBA(Adaptive Multiple Block Ahead) 선반입 기법

(그림 2)는 응용프로그램에서 짧은 시간에 많은 파일 블록들을 요청할 경우 발생하는 OBA 선반입 기법의 문제점을 감소시키고 선반입의 효과를 증대시키는 AMBA 선반입 기법을 보여준다. 응용프로그램에서 충분한 T_int_i 없이 파일 블록들이 사용되는 경우에도 AMBA 선반입 기법에서는 선반입 효과를 얻을 수 있다. 응용프로그램에서 T_int_k 시간에 D_k 및 기존 파일 블록을 사용하고 있는 동안 파일 블록들(D_k, D_{k+1}, ... , D_n)을 선반입한다. 따라서 t_k와 t_{k+1} 사이의 시간 T_int_{k+1}이 하나의 파일 블록을 선반입할 만큼 충분하지 않아도 이전의 여유 시간동안 많은 파일 블록들을 선반입하였으므로, t_{k+1} 시간에 버퍼캐쉬에 선반입되어 있는 파일 블록을 디스크 입출력 없이 사용할 수 있다.

선반입하는 파일 블록의 개수 제한없이 선반입을 수행한다면 T_int_k가 길어질 경우 버퍼캐쉬 내에 존재하는 응용프로그램에서 사용할 가능성이 있는 파일 블록들 혹은 앞서 선반입한 파일 블록들을 교체할 가능성이 있다. 이런 문제를 감소시키기 위해 선반입할 수 있는 최대 파일 블록의 개수를 제한한다.

따라서 버퍼캐쉬 크기가 선반입된 파일 블록들이 캐싱되어 응용프로그램에서 사용하기 전까지 교체되지 않을 정도로 충분히 크고, 선반입된 파일 블록들을 응용프로그램에서 사용하면 시스템 성능은 크게 향상될 것이다.

선반입된 파일 블록들이 버퍼캐쉬에 저장되기 때문에 선반입 기법은 버퍼캐쉬 관리 기법과도 밀접한 관계를 가지게 된다. 파일 블록을 선반입하게 되면, 버퍼캐쉬 내의 기존 파일 블록이 교체되게 된다. 이때 선반입된 파일 블록이 사용되어지지 않는다면 버퍼캐쉬 효율은 떨어질 수 밖에 없다. 이러한 이유로 인해서 선반입 기법은 버퍼캐쉬 관리 기법과 연계해서 고려되어야만 한다.

3.3 버퍼캐쉬 관리 기법

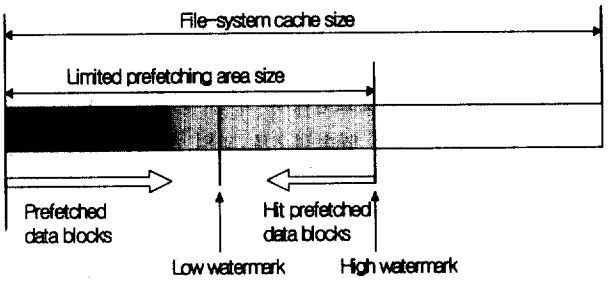
선반입할 파일 블록은 응용프로그램에서 현재 사용 중인 파일 블록의 위치와 버퍼캐쉬 내에 선반입된 파일 블록들에 대한 정보를 바탕으로 결정된다. 응용프로그램에서 필요로 하는 파일 블록과 관계없는 부분을 선반입하는 것은 버퍼캐쉬 내에 불필요한 파일 블록을 캐싱하게 되고, 이로 인해 버퍼캐쉬 내의 사용될 가능성이 있는 파일 블록이 교체될 수 있다. 또한 불필요한 파일 블록의 선반입은 불필요한 디스크 입출력을 발생시켜 시스템 성능에 저하를 가져올 수 있으므로 응용프로그램에서 필요로 하는 파일 블록을 선반입하기 위해 다음에 선반입할 파일 블록의 위치를 결정하는 것은 중요한 일이다. 이미 언급한 것처럼 본 논문에서는 자료를 순차적으로 접근되어지는 응용프로그램으로 대해서 기술한다. AMBA 선반입 기법은 접근형태가 바뀌더라도 예측된 접근형태이면 쉽게 적용할 수 있다. 접근형태 예측은 이미 여러 논문에서 연구되고 있고 많은 결과가 나와 있다.

선반입된 파일 블록들은 같은 버퍼캐쉬 내에 캐싱된다. 버퍼캐쉬의 크기는 한정되어 있으므로 연속적으로 선반입을 하게되면 기존에 캐싱되어 있는 파일 블록들이 교체될 수 있다. 파일 블록의 교체 방법으로는 LRU 캐쉬 정책을 사용할 수 있을 것이다.

물론 응용프로그램에서 요청한 파일 블록이나 선반입하고자 하는 파일 블록이 버퍼캐쉬 내에 존재하게 되면 디스크로부터의 입출력은 없게 된다. 새로운 파일 블록이 선반입되어 버퍼캐쉬에 저장되기 위해서는 가장 오랫동안 사용되지 않았던 파일 블록의 버퍼영역을 비워 주어야 한다.

따라서 선반입된 파일 블록들이 사용되지 않더라도 기존에 캐싱된 파일 블록들이 버려지게 된다. 또한 많은 양의 파일 블록들이 무조건적으로 선반입될 경우, 선반입된 파일 블록들이 버퍼캐쉬 내의 기존 파일 블록들을 교체하게되고, 다시 사용될 파일 블록들까지 교체될 가능성이 커진다. 이는 이중적인 디스크 입출력을 유발하여 시스템의 성능을 저하시킬 수 있다.

이런 문제를 해결하기 위해 시스템의 버퍼캐쉬에서 각 응용에서 최대로 선반입할 수 있는 파일 블록의 개수를 제한한다.



(그림 3) 버퍼캐쉬 영역 내에서 제한된 선반입 영역의 사용

(그림 3)은 버퍼캐쉬영역 내에서 제한된 선반입 영역을 나타낸다.

버퍼캐쉬에서 선반입을 위한 공간의 크기는 버퍼캐쉬의 일정 비율만큼 배정한다. 즉, 응용의 특성에 따라서 선반입을 위한 버퍼캐쉬공간을 동적으로 조정할 수 있게 그 크기가 결정된다. prefetching buffer size는 버퍼캐쉬 내에서 선반입할 수 있는 최대 파일 블록의 개수를 나타낸다. 즉, 선반입된 파일 블록들이 저장될 수 있는 최대 크기를 나타낸다. Prefetched data blocks는 선반입되어 사용되기를 기다리는 파일 블록들이고, Hit prefetched data blocks는 버퍼캐쉬 영역 내의 선반입된 파일 블록들이 응용프로그램에서 참조된 파일 블록들을 말한다. 따라서 참조된 파일 블록 개수만큼 선반입될 수 있는 파일 블록이 늘어나게 된다.

선반입이 중지되었다가 선반입을 보다 효율적으로 하기 위해서 선반입 공간의 크기들의 한계인 high watermark와 재개되는 시점을 나타내는 low watermark를 둔다.

하나의 파일 블록만큼의 여유 공간으로 인해 다시 선반입을 수행하는 것은 시스템에서 과부하로 작용할 수 있기 때문에, 버퍼캐쉬 내의 선반입한 파일 블록들이 캐싱될 충분한 공간적 여유가 생길 때까지 선반입의 수행을 멈춘다.

그리고 선반입 공간의 크기를 정해놓았다고 해서 그 영역에 응용프로그램에서 파일 블록들이 요청될 때 파일 블록들이 캐싱되지 않는다는 것은 아니다. 선반입된 파일 블록들이 어느 순간 캐싱될 수 있는 최대 개수를 나타낼 뿐이다.

선반입된 파일 블록과 응용프로그램의 요청에 의해 캐싱된 파일 블록이 모두 같은 버퍼캐쉬 내에 존재하며 같은 LRU 정책에 영향을 받는다. 이런 이유로 본 논문에서 제안한 선반입 기법은 버퍼캐쉬 내의 참조된 파일 블록들과 선반입된 파일 블록들을 하나의 버퍼캐쉬 내에서 통합적으로 관리한다고 볼 수 있다.

3.4 선반입 블록 위치 결정

선반입 블록 위치란 선반입 수행시 선반입을 시작할 파일 내의 파일 블록의 위치를 말한다. 선반입 수행시 응용프로그램에서 필요로 하지 않는 파일 블록을 선반입하는 것은 불필요한 디스크 입출력을 유발하여 시스템의 성능을 저하시키므로 선반입 블록 위치를 결정하는 것은 중요한 과정이라고 할 수 있다.

하지만 앞으로 선반입 블록 위치 결정에 대한 설명은 파일의 액세스 패턴이 순차적이라는 가정에 바탕을 두고 있지만, 파일의 액세스 패턴이 스트라이드라도 그 패턴이 예측 가능하다면 순차적인 것과 선반입 블록 결정에 있어서 차이가 없다.

임의의 위치로부터 파일 블록을 선반입 한다면 사용되지 않을 파일 블록이나 이미 선반입된 파일 블록을 선반입할 가능성이 커져 불필요한 입출력을 유발하게 되고 버퍼캐쉬의 효율성도 떨어뜨린다. 그래서 AMBA 선반입 기법은 두가지 선반입 블록 위치결정을 가지게 된다. 하나는 응용프로그램이 읽기 요청이 끝나는 파일 블록의 다음 위치이고, 또 하나는 이전의 선반입이 끝난 파일 블록의 다음 위치이다.

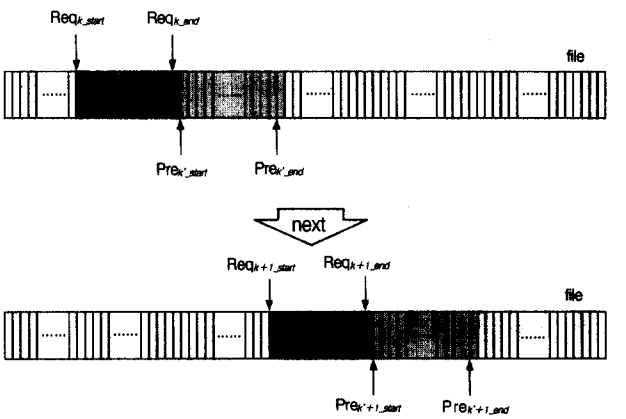
응용프로그램에서 읽기 요청이 끝나는 위치에서 선반입할 경우는 다음과 같다. 응용프로그램에서 파일 블록에 대한 읽기 요청이 있을 경우 요청이 끝나는 파일 블록의 위치로부터 선반입을 하게 되고 응용프로그램이 다음에 사용될 파일 블록을 요청할 경우 디스크 입출력없이 파일 블록을 사용할 수 있게 된다. 이 경우 시스템은 응용프로그램의 파일 블록에 대한 읽기 요청이 있을 때, 요청이 끝나는 파일 블록의 위치를 저장하고, 선반입이 수행될 때 이 위치를 참조하도록 한다.

하지만 응용프로그램에서 읽기 요청을 하고 난 후 다음 읽기 요청을 하기까지의 시간 간격이 길어진다면 응용프로그램이 자신의 요청이 끝나는 파일 블록의 위치를 새로이 갱신하기 전에 선반입이 다시 수행된다. 그래서 이 경우의 파일 블록을 선반입할 때의 위치결정은 이전에 선반입이 끝난 파일 블록의 위치를 기준으로 선반입하여야 한다.

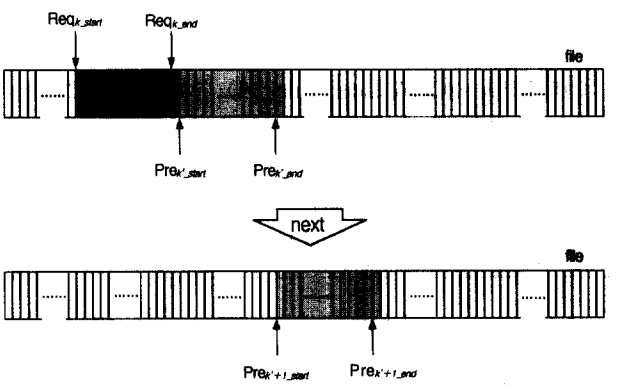
읽기 요청이 끝나고 선반입 요청이 들어오면 기본적으로는 읽기 요청이 끝난 다음 블록에서 선반입 블록 위치가 결정된다. 하지만 전에 선반입한 블록이 많아서 읽기 요청한 블록을 모두 버퍼캐쉬에서 읽은 뒤에도 이전에 선반입한 블록이 남아 있을 수 있다. 이럴 경우에는 이전 선반입이 끝난 블록의 위치를 기준으로 선반입 하여야 한다.

구체적인 예로 (그림 4)는 위에서 설명한 선반입 블록 위치가 파일 내에서 어떻게 표현되는지를 보여준다. 사용된 용어를 정리하면 다음과 같다.

- Req_i : 응용프로그램에서의 i 번째 읽기 요청
- Req_{i_start} : 응용프로그램에서의 i 번째 읽기 요청시 읽어올 파일 블록의 시작위치
- Req_{i_end} : 응용프로그램에서의 i 번째 읽기 요청시 읽어올 파일 블록의 끝위치
- $Pre_{i'}$: i' 번째의 선반입 수행
- $Pre_{i'\_start}$: i' 번째의 선반입이 시작되는 파일 블록의 위치
- $Pre_{i'\_end}$: i' 번째의 선반입이 끝나는 파일 블록의 위치



(a) $Pre_{k'}$ 선반입 후 $Pre_{k'+1}$ 선반입 하기 전에 Req_{k+1} 요청이 있을 경우



(b) $Pre_{k'}$ 선반입 후 Req_{k+1} 요청이 있기 전에 $Pre_{k'+1}$ 의 선반입을 할 경우

(그림 4) 파일 내에서의 선반입 블록 위치 결정

(그림 4)는 응용프로그램이 k 번째 읽기 요청을 하고, 요청이 끝나는 파일 블록의 위치를 시작으로 k' 번째 선반입을 수행하는 경우이다. 이 후 k'+1 번째 선반입이 어디서부터 시작하는지를 보여준다.

(a)는 k' 번째 선반입 수행 후 k'+1번째 선반입 수행 전 k+1 번째 응용프로그램의 읽기 요청이 있을 경우이다. k'+1 번째 선반입 수행 전 k+1 번째 응용프로그램의 읽기 요청이 있었으므로 선반입 블록 위치는 k+1 번째 응용프로그램의 읽기 요청이 끝나는 파일 블록의 위치가 k'+1 번째 선반입 수행의 시작 위치가 되는 것이다($Pre_{k'+1\_start} = Req_{k+1\_end}$).

(b)는 k' 번째 선반입 수행 후 k+1 번째 응용프로그램의 읽기 요청이 있기 전에 k'+1 번째 선반입이 수행되는 경우이다. k' 번째 선반입 수행시 k 번째 응용프로그램이 읽기 요청이 끝나는 지점을 이미 참조하였으므로($Pre_{k'\_start} = Req_{k\_end}$), 이전 선반입이 끝나는 위치를 기준으로 선반입을 수행한다($Pre_{k'+1\_start} = Pre_{k'\_end}$). 이는 k 번째와 k+1 번째간의 응용프로그램의 요청 시간 간격이 길어질 경우에 적용된다.

하지만 무조건적인 선반입은 버퍼캐쉬 내에 존재하는 다른 파일 블록들을 교체할 가능성을 높이기 때문에 응용프로그램에서 읽기 요청을 하고 난 후 다음 읽기 요청을 하기까지의 시간 간격이 길어질수록 버퍼캐쉬의 효율성에 문제가 생길 것이다. 따라서 파일을 액세스하는 패턴마다 버퍼캐쉬 내에 선반입되는 파일 블록들의 개수를 제한하는 방법으로 이 문제를 완화할 수 있다. 선반입되는 블록들의 개수를 제한하는 방법은 버퍼캐쉬 내에 선반입된 파일 블록들이 캐싱될 공간인 선반입 영역을 미리 정해 놓음으로써 가능하다.

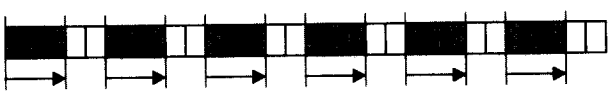
4. 실험 및 결과

실험은 AMBA 선반입 기법과 OBA 선반입 기법을 PFSL[8]에 적용하여 각각의 성능을 비교하였다. 적용된 응용 프로그램은 기존에 연구되었던 파일 액세스 패턴을 기반으로 하여 그와 유사한 패턴으로 접근이 되도록 하였다.

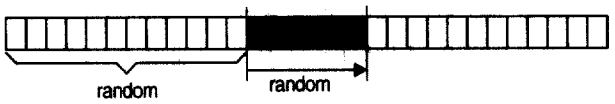
4.1 적용된 파일 액세스 패턴

CHARISMA(**CHA**Racterizing **I/O** in **S**cientific **M**ulti-processor **A**pplications) 프로젝트와 Pablo 프로젝트에서 파일 액세스 패턴에 대한 분석이 이루어졌다[1, 7].

(a) Pattern 1 (Sequential)



(b) Pattern 2 (Simple strided)



(c) Pattern 3 (Random)

(그림 5) 연구 결과를 바탕으로 정의된 파일 액세스 패턴

이 연구 결과를 바탕으로 구현된 시스템에 적용할 패턴을 정의하였으며, (그림 5)는 정의한 패턴의 형태이다. (a)는 일반적인 파일 시스템을 포함하여 대부분의 응용 프로그램 및 멀티미디어 응용에서 발견되는 형태이다. (b)는 병렬 처리 시스템 상의 과학 기술 계산 응용에서 주로 나타나는 형태이다. (c)는 (a)와 (b)를 제외한 나머지 모든 패턴을 포함하는 형태로 정의하였다.

4.2 실험 환경 및 방법

응용프로그램은 파일서버와 같은 노드에서 실행이 되고, 파일서버와 블록서버는 각각 다른 노드에서 동작하도록 하였다. 네트워크의 구성은 실험에 사용된 노드들만의 지역 네트워크를 구성하였다. 다음은 실험 환경과 관계된 세부 사항이다.

- 운영체제 : Linux kernel version 2.2.12
- 파일서버 노드의 수 : 1대
- 블록서버 노드의 수 : 3대
- Network Interface Card : 1Gbps, Clan
- 하나의 파일 블록의 크기 : 8 KBytes
- 파일에 대해 전체적으로 액세스 된 크기 : 약 100 MBytes
- 실험 측정값 : 한 패턴에 대해 4 번의 실험을 평균한 값

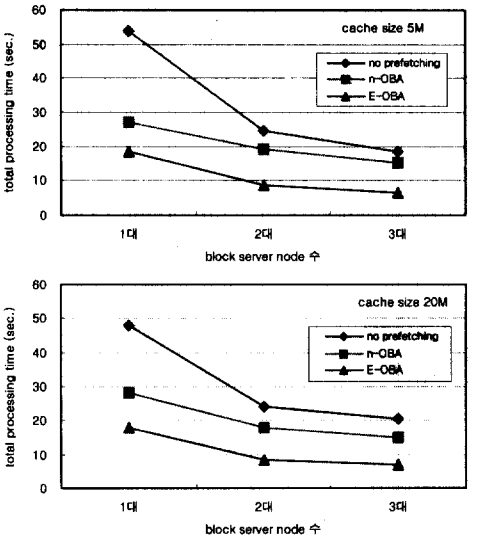
OBA 선반입 기법을 적용한 시스템과 AMBA 선반입 기법을 적용한 시스템 상에서 응용프로그램을 실행하여 제안한 AMBA 선반입 기법의 성능 개선 정도를 측정하였다. 실험에 사용된 OBA 선반입 기법은 응용프로그램의 요청이 있은 후 하나의 파일 블록만을 선반입하는 것이 아니라 응용프로그램에서 한번에 요청하는 파일 블록들의 개수만큼 선반입을 하도록 하였다. 즉, OBA 선반입 기법보다 개선된 n-OBA(n= 64) 선반입 기법을 적용하였다.

AMBA 선반입 기법의 제한된 선반입 영역은 전체 버퍼캐쉬 크기의 90%로 정하였다. 이는 되도록 많은 파일 블록들의 선반입을 보장해 주고, 한번 선반입된 파일 블록들의 재교체 가능성을 줄여준다.

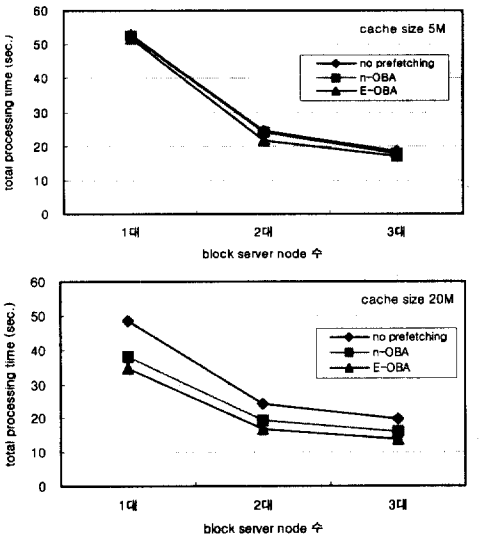
4.3 실험 결과

시스템 성능은 실험에 사용된 파일의 열기부터 닫기까지 읽기 수행을 행한 시간(total processing time)으로 나타내었다. 또한 각각의 파일 액세스 패턴별로 구분하여 버퍼캐쉬의 크기를 변화시켜 가면서 선반입에 의한 영향을 알아보았다.

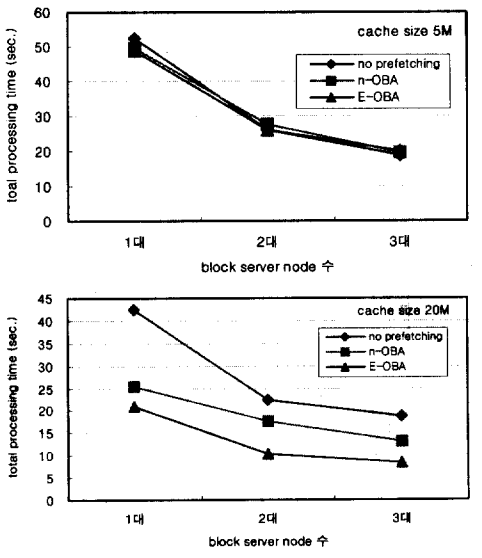
(그림 6), (그림 7) 그리고 (그림 8)은 기존의 OBA 선반입 기법과 제안된 AMBA 선반입 기법을 적용했을 때의 각각의 패턴들에 대한 시스템 성능을 나타낸다.



(그림 6) pattern 1에서의 시스템 수행 시간



(그림 7) pattern 2에서의 시스템 수행 시간



(그림 8) pattern 3에서의 시스템 수행 시간

전체적으로 선반입을 적용하였을 경우가 적용하지 않았을 경우보다 시스템 성능이 개선된다. 하지만 사용되지 않을 파일 블록들을 선반입할 가능성이 있는 pattern 2와 pattern 3의 경우, 적은 크기의 버퍼캐쉬에서는 선반입에 의한 효과가 거의 없으며, 시스템에 과부하를 주는 경우도 발생한다. 이는 응용프로그램에서의 캐싱되는 파일 블록들과 선반입되는 파일 블록들이 버퍼캐쉬 내에 충분히 저장되지 못하므로 빈번한 교체 현상이 발생할 수 있고, 이로 인해 필요이상의 디스크 입출력이 유발될 수 있기 때문이다.

가장 큰 성능 개선은 순차적인 패턴인 pattern 1에서 이루어진다. 이 패턴은 선반입의 영향을 가장 많이 받는 형태로써, OBA 선반입 기법이 적용된 시스템은 선반입하지 않은 시스템에서보다 15~50% 정도 성능이 향상되었고, AMBA 선반입 기법이 적용된 시스템은 선반입하지 않은 시스템에서보다 60% 정도 성능이 향상되었다. 또한 AMBA 선반입 기법이 적용된 시스템은 OBA 기법이 적용된 시스템에 비해 30~40% 정도 성능이 향상되었다.

Pattern 2와 pattern 3에서도 버퍼캐쉬 크기가 충분히 클 경우 선반입에 의한 성능 향상이 있으며, AMBA 선반입 기법이 적용된 시스템은 OBA 선반입 기법이 적용된 시스템에 비해 10~30% 정도 성능이 향상되었다. pattern 2와 pattern 3 역시 부분적으로는 액세스 형태가 순차적이므로 선반입에 의한 성능 개선이 있으며, pattern 1에서 보다 사용되지 않을 파일 블록들이 선반입될 가능성이 크므로 pattern 1보다는 개선 정도가 떨어진다.

모든 경우에 대해서 공통적인 개선 사항은 입출력 노드 즉, 블록서버 증가에 따른 성능 개선이다. 이는 디스크에서 발생하는 데이터 병목을 다수의 블록서버로 분산시킴으로써 디스크에서 오는 시간 지연을 감소시켜주기 때문이다.

5. 결 론

멀티미디어 서비스 및 병렬 과학 기술 계산 응용 등이 점차 많아지는 시점에서 사용자의 요청을 빠르게 서비스해 주는 문제의 해결책으로 클러스터링이 제시되었다. 그 중 병렬 파일 시스템은 클러스터 시스템을 이루는 각각의 시스템들의 디스크 입출력 병목 문제를 해결하는 소프트웨어적인 방법이다. 본 논문에서는 병렬 파일 시스템의 성능을 향상시키는 선반입 기법을 제안하였다. 제안된 선반입 기법은 AMBA 선반입 기법으로 OBA 선반입 기법의 개념을 확장한 것이다. OBA 선반입 기법은 응용프로그램의 요청이 있은 후 일정한 양의 파일 블록들을 선반입하지만, 제안한 AMBA 선반입 기법은 응용프로그램의 요청에 대해 비동기적으로 파일 블록들을 선반입하는 것으로 선반입되는 파일 블록의 양은 시스템 상황에 따라 동적으로 변한다. 이는 버퍼캐쉬 내에 선반입될 수 있는 영역을 제한함으로써 가능하다.

병렬 파일 시스템의 응용에 자주 나타나는 파일 액세스 패턴으로 실험한 결과 제안된 AMBA 선반입 기법이 적용된 시스템의 성능이 기존 OBA 선반입 기법이 적용된 시스템의 성능에 비해 10~40% 정도 개선되었다. 특히, 순차적인 패턴의 경우 성능 개선 정도가 30~40% 정도로 안정적이면서 높게 나타났다. 또한 버퍼캐쉬 크기가 작을 경우 순차적인 패턴 이외의 패턴에서는 시스템의 성능이 개선되지 않지만, 버퍼캐쉬 크기가 클 경우 10~30% 정도 시스템 성능이 개선되었다.

또한 버퍼캐쉬 크기가 클 경우 선반입되는 파일 블록의 양이 많을수록 시스템의 성능이 개선된다. 버퍼캐쉬 크기가 크기 때문에 파일 블록들이 교체될 가능성이 줄어들고 그만큼 버퍼캐쉬에 존재하는 시간이 길어져 사용될 확률이 많기 때문이다.

순차적인 패턴 및 간단한 스트라이드 패턴은 액세스되는 형태가 규칙적이므로 선반입을 하지 않았을 경우에 비해 각각 40~60%, 18~26% 정도 성능이 개선되었고, 성능 개선의 정도도 안정적이다. 이에 비해 불규칙적인 액세스 패턴의 경우에는 8~50% 정도로, 시스템 성능 개선이 안정적이지 못하다.

시스템이 확장되었을 경우 그와 비례해서 시스템의 성능이 개선된다. 따라서 큰 파일을 다루는 복잡한 과학 계산 응용 및 멀티미디어 서비스에서는 서비스 시간을 단축시키기 위해 병렬 파일 시스템의 사용이 이루어져야 하며 이와 함께 AMBA 선반입 기법은 병렬 파일 시스템의 성능을 효과적으로 개선시켜 줌을 보여준다.

참 고 문 헌

[1] A. Purakayastha, C. S. Ellis, D. Kotz, N. Nieuwejaar and M. Best, "Characterizing Parallel File-access Patterns on a Large-scale Multiprocessor," In Proc. of the Ninth International Parallel Processing Symposium, pp.165-172, April, 1995.

[2] A. Purakayastha, "Chracterizing and Optimizing and Parallel File Systems," Dissertation of Duke University, Durham, N. H. pp.1-10, June, 1996.

[3] R. H. Patterson and G. A. Gibson, "Exposing I/O concurrency with informed prefetching," In Proc. Third International Conf. on Parallel and Distributed Information Systems, pp.7-16, September, 1994.

[4] K. M. Curewitz, P. Kristian and J. S. Vitter, "Practical prefetching via data compression," In Proc. of the SIGMOD Management of Data, pp.257-266, ACM Press, May, 1993.

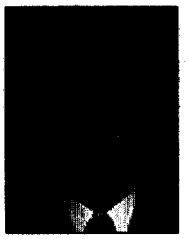
[5] T. Cartes, "Cooperative Caching and Prefetching in Parallel / Distributed File Systems," Ph. D Thesis, Universitat Politecnica de Catalunya, 1997.

[6] P. Cao, E. W. Felten, A. R. Karlin and K. Li, "A study of integrated prefetching and caching strategies," In Proc. 1995 ACM SIGMETRICS, pp.188-197, May, 1995.

[7] E. Smirni and D. A. Reed, "Workload Characterization of Input/Output Intensive Parallel Applications," In Proc. of the Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Springer-Verlag Lecture Notes in Computer Science, pp.169-180, Vol.1245, June, 1997.

[8] J. H. Cho, C. Y. Kim and D. W. Seo, "Parallel File System Using Dual Caches Scheme and Prefetching," The 2000 International Conference on Parallel/Distributed Processing Techniques and Application (PDPTA2000), June, 2000.

[9] 조성제, "페이지 선반입시 메모리 오염 감소 정책과 선반입 버퍼 할당정책", Journal of KISS(A) : Computer Systems and Theory Vol.24, No.11, pp.1152-1161, 1226-2315, NOVEMBER, 1997.



임 재 덕

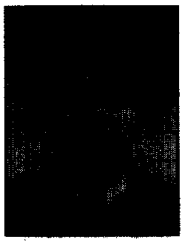
e-mail : jdscol92@etri.re.kr

1999년 경북대학교 전자공학과 졸업(학사)

2001년 경북대학교 대학원 전자공학과 졸업(석사)

2000년~현재 한국전자통신연구원 연구원

관심분야 : 병렬 처리, 시스템 보안, 정보보호, 접근제어



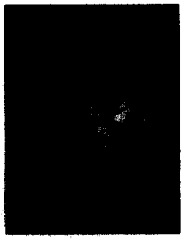
황 보 준 형

e-mail : bluesky@palgong.knu.ac.kr

2000년 경북대학교 전자전기공학부 졸업(학사)

2000년~현재 경북대학교 전자공학과 석사과정 재학중

관심분야 : 병렬처리, 클러스터 컴퓨팅, 운영체제, 시스템 보안



고 광 식

e-mail : kskoh@ee.knu.ac.kr

1980년 경북대 전자공학과 전자공학(학사)

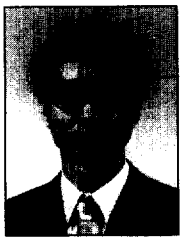
1983년 한국과학기술원 전자공학과 전자공학(석사)

1985년~1987년 경북대학교 전자공학과 전임강사

1988년~1992년 경북대학교 전자공학과 조교수

1993년~현재 경북대학교 전자공학과 부교수

관심분야 : 디지탈시스템설계, 신호처리, 컴퓨터비젼



서 대 화

e-mail : dwseo@ee.knu.ac.kr

1981년 경북대학교 전자공학과(학사)

1983년 한국과학기술원 전산학과(석사)

1993년 한국과학기술원 전산학과(박사)

1981년~1995년 한국전자통신연구소 시스템 S/W 연구실 근무

1998년~1999년 University of California Irvine 연구 교수

1995년~현재 경북대학교 공과대학 전자전기공학부 부교수

관심분야 : 병렬분산처리, 운영체제, 병렬처리, 컴퓨터 구조