반복성을 고려한 파일 액세스 패턴 수집 기법

(File Access Pattern Collection Scheme based on Repetitiveness)

황보 준 형 † 석 성 우 †† 서 대 화 †††

(Jun-Hyoung Hwang-Bo) (Song-Woo Sok) (Dae-Wha Seo)

요 약 본 논문에서는 액세스 패턴의 반복성을 이용하여 비교적 적은 메모리 공간을 사용하는 SIC (Size-Interval-Count) 선반입 기법을 제안한다. 최근에 연구되어진 지식기반의 선반입 기법은 응용프로그램의 액세스를 예측하여 정확한 선반입을 수행하는 기법이다. 이들 기법은 응용프로그램의 액세스 패턴을 기록하고, 기록된 액세스 패턴정보를 이용하여 다음에 요청될 블록을 예측하게 된다. 하지만 이 기법은 많은 메모리 공간의 사용을 필요로 한다. 따라서 제안된 선반입 기법에서는 "SIC 액세스 패턴 정보"를 이용하여 반복적인 액세스 패턴을 효율적으로 저장하고, 이를 이용하여 응용프로그램의 다음에 요청될 블록을 정확하게 예측한다. 본 논문의 선반입 기법은 일반 파일시스템에 비해 최고 40%의 응답속도 향상을 가져오며, 기존의 지식기반 선반입 기법에 비해 뛰어난 메모리 효율성을 보여준다.

Abstract This paper presents the SIC(Size-Interval-Count) prefetching scheme that can record the file access patterns of applications within a relatively small space of memory based on the repetitiveness of the file access patterns. Several knowledge-based prefetching methods were recently introduced, which includes high correctness in predicting future accesses of applications. They records the access patterns of applications and uses recorded access pattern information to predict which blocks will be requested next. Yet, these methods require to much memory space. Accordingly, the proposed method then uses the recorded file access patterns, referred to as "SIC access pattern information", to correctly predict the future accesses of the applications. The proposed prefetching method improved the response time by about 40% compared to the general file system and showed remarkable memory efficiency compared to the previously knowledge-based prefetching methods.

1. 서 론

하드디스크 등의 저장장치의 대용량화와 멀티미디어 데이타를 포함하는 대용량 파일이 차지하는 비중이 커짐에 따라 평균적인 파일의 크기가 크게 증가하고 있다. 또한, 네트워크 대역폭의 확대와 컴퓨터 시스템의 계산능력이 증가함에 따라 파일 입출력 요구량이 증가하고 있다[1].

멀티미디어 데이타와 같은 대용량 데이타들은 재사용성이 거의 없기 때문에 한번만 요청되고 더 이상 요청되지 않는 데이타 블록이 대부분이다. 이에 따라 버퍼 캐쉬의 적중률이 감소하고 있다. 이러한 액세스 패턴 하에서 버퍼 캐시 적중률을 향상시키기 위해서는 선반입의 중요성이 커지고 있다.

선반입은 응용프로그램이 데이타를 액세스하기 전에 응용프로그램이 요청할 것으로 예측되는 데이타 블록을 미리 디스크에서 읽어서 버퍼 캐쉬에 저장함으로써 버퍼 캐쉬의 적중률을 향상시킨다. 그리고 응용 프로그램의 읽기 요청을 디스크에서가 아닌 버퍼 캐쉬로부터 해당 데이타 블록을 읽어옴으로 해서 전체 파일 시스템의 응답속도를 향상시킬 수 있다.

하지만, 선반입 역시 디스크 입출력 대역폭을 차지하기 때문에, 선반입한 데이타 블록이 사용되지 않는다면 디스크 입출력 대역폭을 낭비하는 것이 된다. 또한 선반입한 데이타 블록을 저장하기 위해 캐시 버퍼에서 제거된 데

† 학생회원 : 경상대학교 전자공학과
bluesky@palgong.knu.ac.kr
†† 정 회 원 : 한국전자통신연구원 컴퓨터 소프트웨어 연구소 연구원
swsok@etri.re.kr
††† 종신회원 : 경북대학교 전자전기공학부 교수
dwseo@ee.knu.ac.kr
논문접수 : 2001년 4월 25일
심사완료 : 2001년 9월 10일

이타 블록에 대한 액세스가 발생한다면 이를 다시 디스크에서 읽어와야 하므로 이중으로 디스크 입출력 대역폭을 낭비하게 될 뿐만 아니라 캐시 적중률을 저하시켜 파일 시스템의 응답속도가 떨어지게 된다. 따라서 미래에 사용될 데이타 블록에 대한 예측 정확도가 높지 않다면 선반입은 오히려 파일 시스템의 성능을 저하시키게 된다.

이러한 정확하지 못한 선반입에 의해 성능이 저하되는 것을 해결하기 위해 응용프로그램의 액세스를 예측하여 선반입을 수행하기 위한 연구가 있어 왔다[2][7] [8]. 본 논문에서는 지능적 액세스 패턴 예측 방법의 관점에서 새로운 선반입 기법인 SIC(Size-Interval-Count) 기법을 제안한다. 본 논문의 SIC기법은 일반적인 응용프로그램의 액세스 패턴은 이전에 나타난 액세스 패턴을 다시 나타낼 가능성이 높다는 가정 하에서 이전에 나타난 액세스 패턴을 사용하여 선반입을 수행한다.

SIC기법은 응용프로그램의 파일 액세스를 시간 순서대로 기록한 액세스 패턴 정보를 현재의 액세스 패턴과 비교하여 앞으로의 액세스 패턴을 예측하고 이에 맞추어 선반입을 수행한다. 따라서, 실행 시마다 액세스 패턴이 변화하는 데이타베이스와 같은 응용은 제안하는 선반입 기법에서는 선반입을 수행하지 않는다.

SIC기법은 이러한 변화가 심한 액세스 패턴이 아니라 액세스 패턴에 규칙성이 있는 파일에 대해 선반입을 수행한다. 이러한 규칙성이 있는 액세스는 일반적인 UNIX 파일 시스템[3]이나 과학 기술 계산을 위한 병렬처리 시스템[4][5] 등 다수의 시스템에서 평균적으로 80% 이상의 액세스를 차지한다. 따라서, 데이타베이스 등의 특별한 응용이 아닌 일반적인 파일 입출력 요청에 대해서는 규칙성이 있는 액세스 패턴에 대해서 선반입 할 수 있다면 충분한 성능향상을 이룰 수 있게 된다.

본 논문은 서론에 이어 2장에서는 파일 액세스 패턴 연구 및 기존의 선반입 정책 연구에 대해 알아보고 기존 연구의 문제점을 고찰한다. 3장에서는 본 논문에서 제안하는 새로운 선반입 기법인 SIC기법에 대해 알아보고 4장에서는 SIC기법의 패턴 수집방법에 대해서 기술한다. 5장에서는 수집된 패턴에 따라 선반입 수행방법에 대해 설명하고, 6장에서는 구현된 선반입 시스템의 성능을 평가하기 위해 기존의 파일 액세스 패턴 연구에서 알려진 액세스 패턴을 사용해서 실험한 결과를 제시한다. 마지막으로 7장에서는 실험 결과를 바탕으로 결론을 내리고 앞으로의 과제를 제시한다.

2. 관련 연구

기존 선반입 정책에는 OBL(One Block Lookahead), IBL(Infinite Blocks Lookahead)기법등이 있다[6]. 이 선반입 기법들에 대해 알아보면 다음과 같다.

OBL은 가장 대표적인 선반입 정책으로서 OBA(One Block Ahead)라고 불리기도 한다. OBL 선반입 정책은 현재 읽기가 요청된 블럭의 다음 블럭을 선반입하기 때문에 파일의 블럭을 차례대로 액세스하는 순차적인 액세스가 많은 경우에 효과적이다.

IBL은 OBL을 확장한 선반입 정책으로 읽기가 요청된 블럭의 다음 블럭부터 순차적으로, 자원이 허락하는 한도 내에서 최대한 선반입을 수행한다. IBL은 순차적인 액세스가 대부분인 경우, 그리고 입출력이 많은 응용프로그램의 경우에 효과적이다. 입출력 요구가 많은 상황에서는 한번에 선반입하는 양을 늘임으로써 저장장치의 지연시간에 의한 오버헤드를 줄일 수 있다. IBL는 순차적 액세스가 많은 경우, 그리고 입출력 요구가 많은 시스템에서 저장장치의 지연시간에 의한 오버헤드를 줄임으로써 OBL보다 높은 성능을 낸다.

이들 선반입 기법에서는 대부분의 액세스가 순차적 액세스 패턴이거나 단순 스트라이드 액세스 패턴에서는 충분한 성능향상을 기대할 수 있다.

하지만, 현재의 상황은 응용 프로그램의 액세스 패턴이 이전에 비해 복잡해지고 있으며, 이러한 액세스 패턴에 대한 선반입의 필요성도 증가하고 있다[2]. 따라서 응용프로그램의 액세스 패턴을 사용하여 선반입을 수행함으로써 선반입의 정확성을 향상시키고 선반입을 적용할 수 있는 액세스 패턴의 범위를 확대하기 위한 연구가 진행되고 있다.

응용프로그램의 액세스 패턴을 사용해서 선반입의 정확성을 향상시키는 방법에는 크게 두 가지 방법이 있다. 하나는 응용프로그램이 특별한 API를 사용하여 앞으로 사용하게 될 블럭에 대한 정보를 파일 시스템에 제공함으로써 파일 시스템이 이 정보를 사용하여 선반입을 수행하는 '힌트기반 선반입 기법'이고, 또 다른 하나는 파일 시스템에서 응용프로그램의 액세스 패턴을 분석하여 다음에 액세스할 블럭을 예측하고 이에 따라 선반입을 수행하는 '지능적 액세스 패턴 예측 방법'이다[9][10].

힌트기반 선반입 기법을 적용하기 위해서는 파일 시스템이 응용프로그램의 힌트를 전달받기 위한 특별한 API를 제공하여야 하고, 응용프로그래머는 이 API를 사용해서 응용프로그램을 작성해야만 한다. 또한 응용프로그래머는 작성하는 응용프로그램의 저 수준 입출력 패턴에 대해 알고 있어야 할 뿐만 아니라 미래의 입출력 패턴을 예측하는 루틴을 작성하여 추가하여야 한다.

지능적 액세스 패턴 예측 방법도 분류에 포함되는 액

세스 패턴에 대해서만 적용이 가능하고, 정의된 액세스 패턴 종류가 많을수록 복잡한 계산이 필요하기 때문에 계산 부하가 증가한다. 또한 파일 액세스가 시작된 이후, 일정 시간동안의 액세스 패턴을 관찰해서 패턴을 발견해야만 이후의 액세스 패턴을 예측할 수 있기 때문에, 파일 액세스가 시작된 직후 액세스 패턴의 변화가 발생한 직후에는 성능이 저하된다.

기존의 방법에서는 액세스 패턴이 가지는 반복성에 상관없이 액세스 패턴 정보를 기록하였기 때문에 액세스 패턴 정보의 길이가 파일의 길이에 비례해서 커지게 된다. 따라서 점점 파일의 크기가 커지고 있는 현재의 상황에서는 액세스 패턴정보가 차지하는 메모리 사용량이 증가하게 된다. 이러한 문제점을 해결하기 위해서는 액세스 패턴 정보의 크기를 줄일 수 있는 방법이 필요하다.

다음 장에서는 이런 문제를 해결하기 위해 파일의 액세스가 반복적인 패턴일 때는 액세스 패턴의 정보를 압축하여 저장하는 Size-Interval-Count 액세스 패턴 기반의 선반입 기법인 SIC를 제시한다.

3. SIC 기법

SIC(Size-Interval-Count)는 지식 기반 선반입 방법을 기반으로 한다. 즉, 어떤 파일을 열어서 닫을 때까지의 읽기 액세스를 시간 순서대로 기록해서 저장해두고, 선반입 시에는 과거에 기록된 액세스 순서 정보와 현재의 액세스를 비교해서 다음에 액세스할 데이타 블록을 예측한다. 이 SIC 선반입 기법은 기존의 선반입 기법에 비해 파일 액세스 패턴을 근거로 하는 좀더 정확한 선반입으로 응용프로그램의 응답속도를 높여준다. 그리고 액세스 패턴의 반복성을 이용하여 액세스 패턴 저장시 정보를 본 논문에서 제공하는 압축기법을 이용하여 저장하는 방법으로 시스템의 메모리 효율성을 높여준다. SIC 기법은 특정한 패턴을 가지지 않는 복잡하고 랜덤한 액세스 패턴에 대해서는 패턴 저장을 하지 않는다. 특정한 패턴을 가지지 않는 액세스 패턴까지 저장하여 파일 시스템에서 관리하는 액세스 패턴이 많아진다면 시스템의 과부하가 되어 시스템 성능이 저하될 수 도 있다.

액세스의 반복성이란 그림 1 과 같이 파일 내에서 읽은 부분과 읽지 않고 건너 뛴 부분이 일정한 패턴을 이루고 이 패턴이 반복적으로 발생하는 것을 말한다. 기존의 파일 액세스 패턴 연구에 따르면 전체 파일 액세스 중에서 이러한 반복적인 액세스가 90%이상인 것으로 조사되었다[3-5].

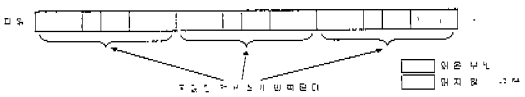

그림 1 액세스의 반복

SIC에서는 파일에 대한 액세스를 기록할 때, 전체 액세스를 기록하지 않고, 반복단위를 구성하는 읽은 부분의 크기와 읽은 부분과 다음 읽은 부분 사이의 간격, 그리고 이 반복단위의 반복회수를 기록함으로써 전체 액세스를 기록하는 것에 비해 액세스 패턴 정보의 크기를 줄여 시스템의 메모리 사용의 효율성을 높여준다.

다음은 SIC 선반입 기법에서 액세스 패턴을 저장하는데 사용하는 액세스 패턴 정보의 구성 인자에 대한 설명이다.

- Size : 패턴에서 읽은 부분의 크기
- Interval : 읽은 부분 사이의 간격(읽지 않은 부분의 크기)
- Count : 패턴의 반복회수

선반입 수행 시에는 이와 같이 기록된 SIC 액세스 패턴 정보를 응용프로그램의 액세스와 비교해서 다음에 발생할 액세스를 예측한다. 그림 2는 SIC 선반입 기법에서 다음에 발생할 액세스를 예측하는 방법을 보여주고 있다. 과거에 발생한 액세스 패턴 정보와 현재 응용프로그램이 액세스하는 것을 비교하여 현재 요청된 액세스에 뒤이어서 발생할 액세스를 과거에 발생한 액세스와 동일할 것이라고 가정하고 이 액세스들이 응용프로그램에서 요청하기 전에 선반입하여 캐시 버퍼에 저장하는 것이다.

응용프로그램의 액세스가 과거와 동일하다면 이러한 예측방법은 캐시 적중률을 100%에 가깝게 향상시킬 수 있고, 비슷하더라도 어느 정도의 성능 향상이 있게 된다. 또한, 과거의 액세스와 현재의 액세스가 다르다면 이러한 예측방법은 선반입의 정확성을 저하시키게 되므로, 계속 현재의 액세스와 과거의 액세스를 비교하여 일치하는지 확인하고, 일치하지 않는다면 선반입을 중지시킴으로써 파일 입출력 성능이 저하되는 것을 막는다.

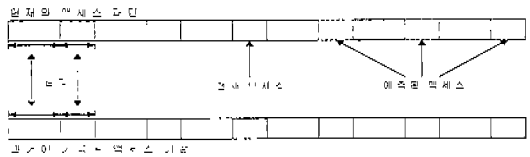

그림 2 SIC 선반입 기법의 액세스 예측

SIC 선반입 기법은 지식 기반 선반입 방법의 특성에 따라, 두 부분으로 구성된다. 액세스 패턴 수집 부분과 선반입 수행 부분이 그것이다. 그림 3은 SIC 선반입 기법을 구성을 나타내고 있다.

액세스 패턴 수집 부분은 응용프로그램의 읽기 액세스가 발생할 때, 선반입 수행부분에서 사용할 액세스 패턴 정보를 수집하고 저장하는 작업이고, 선반입 수행 부분은 저장된 액세스 패턴 정보를 사용해서 선반입할 데이타 블록을 결정하고 선반입을 수행하는 작업이다.

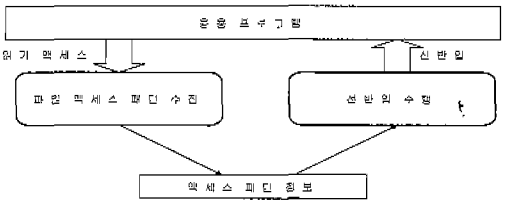

그림 3 SIC 선반입 기법의 액세스 패턴 수집과 선반입 수행

4. SIC 액세스 패턴 수집

파일 액세스 패턴 수집이란 응용프로그램의 액세스를 발생 순서에 따라 기록하여 선반입을 수행하기 위해 필요한 액세스 패턴 정보의 형태로 만드는 작업을 말한다.

SIC 액세스 패턴 정보는 패턴 테이블(PT, pattern table)의 패턴 테이블 엔트리(PTE, pattern table entry) 형태로 기록되게 되는데 PT안의 PTE 순서에 따라 시간적인 순서를 나타낸다. 즉, 앞부분에 기록된 PTE가 뒷부분에 기록된 PTE에 비해 시간적으로 앞서 발생한 것이다.

4.1 실제 액세스와 SIC 액세스 패턴 정보의 관계

그림 4는 액세스 인자와 SIC 액세스 패턴 정보의 각 인자의 관계를 보여주고 있다. 다음은 각 용어의 설명이다.

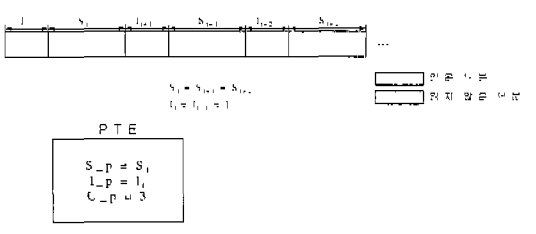

그림 4 실제 액세스 인자와 SIC 액세스 패턴 정보 인자의 관계

그림 4에서 이용된 S_t는 t시간에서의 읽기 요청 크기를 나타내며, l_t는 t-1시간에서의 읽기 요청이 수행 종료되었을 때의 오프셋과 t시간에서의 읽기 요청이 시작될 때의 오프셋간의 거리, 즉 읽기 요청간의 거리를 나타낸다. S_p_i는 i번째 액세스 패턴의 읽기 크기를 나타내고, l_p_i와 C_p_i는 각각 i번째의 액세스 패턴의 읽기 액세스간의 거리와 액세스 패턴의 반복회수를 의미한다.

그림 4에서 보이는 패턴은 단순 스트라이드 패턴이다. l_t와 S_t가 하나의 패턴을 이루고 있고 이 패턴이 3번 반복되고 있다. 이런 액세스 패턴에 대한 SIC 액세스 패턴 정보는 그림의 PTE에서와 같이 읽기 크기는 S_t가 되고 액세스간 간격은 l_t가 된다. 반복회수는 그림에 보이는 것은 3번이므로 3으로 기록되어 있다. 이 패턴이 계속 반복된다면 반복회수는 그 반복회수만큼 증가하게 된다.

이와 같은 하나의 패턴이 파일의 처음부터 끝까지 반복해서 발생한다면 SIC 액세스 패턴 정보는 하나의 PTE로 모든 액세스를 나타내게된다.

4.2 액세스 패턴 정보 표현 방법

SIC액세스 패턴 수집의 방법은 크게 단순 반복 액세스 패턴, 이중 반복 액세스 패턴의 두가지 형태로 나타난다. 이중 반복 액세스 보다 더 복잡한 형태의 액세스 패턴 즉, 반복성을 가지지 않는 패턴과 같은 형태의 액세스 패턴을 가지는 응용프로그램의 존재는 그리 많지 않다. 또한 삼중, 사중 반복 액세스 패턴을 압축해서 기록하는 것은 오히려 오버헤드가 되어 액세스 패턴 수집의 성능을 저하시킬 우려가 있어 이중 반복 액세스 패턴을 기록하는 것만으로도 충분할 것으로 가정한다.

4.2.1 단순 반복 액세스 패턴

단순 반복 액세스는 반복되는 패턴이 하나의 읽기 크기와 하나의 액세스간 간격으로 표현 가능한 것을 말한다.

순차적 액세스 패턴은 파일의 처음부터 끝까지 일정한 읽기 크기를 가지고 있고 읽기 요청간의 간격은 0이

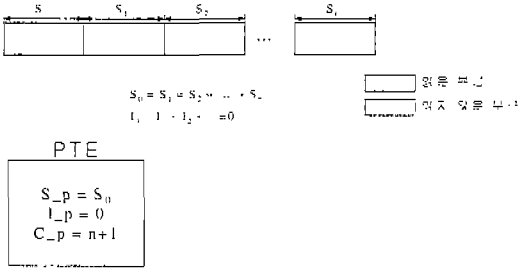

그림 5 순차적 액세스 패턴에 대한 SIC 액세스 패턴 정보

다. 따라서 그림 5와 같이 하나의 PTE로 표현되게 된다. 여기서 n+1은 액세스 패턴의 반복 회수이다.

그리고 단순 스트라이드 패턴은 앞의 그림 4에서 보이는 바와 같이 표현된다. 순차적 액세스 패턴과 동일하게 하나의 PTE로 표현하게 되지만, 순차적 액세스 패턴과 다른 점은 읽기 요청간의 간격이 0이 아니라는 점이다.

4.2.2 이중 반복 액세스 패턴

하나의 읽기 크기와 하나의 읽기 요청간의 거리로 표현할 수 없는 좀더 복잡한 액세스 패턴이 반복적으로 발생하는 경우가 있다. 그림 1의 액세스 패턴은 이러한 복잡한 액세스 패턴의 한 예이다.

이러한 복합적인 액세스 패턴은 SIC 액세스 패턴 정보에서는 몇 개의 PTE가 그룹을 형성하여 하나의 패턴을 이루고 그 패턴이 반복성을 보이게 된다. 그림 6은 복합적인 액세스 패턴에 대해 PTE가 반복적으로 나타나는 것을 보인 것이다.

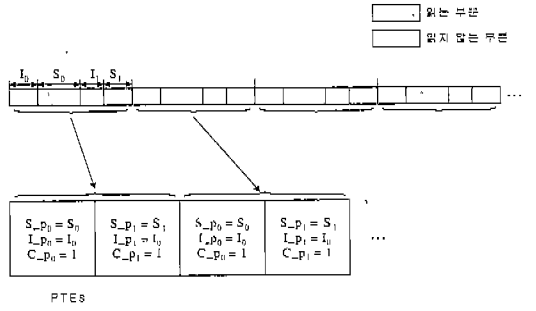

그림 6 복합적인 반복 액세스 패턴과 SIC 액세스 패턴 정보

이러한 복합적인 반복 액세스 패턴을 압축하기 위해 SIC 액세스 패턴 정보에 대한 압축을 사용한다. 그림 7은 그림 6에서 보인 PTE를 압축하는 방법을 보여준다.

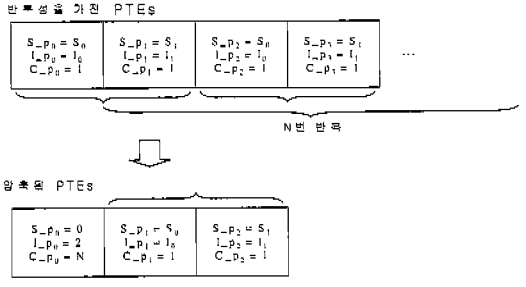

그림 7 반복성이 있는 PTEs 압축

반복성을 가지는 PTE를 압축하는 방법은 PTE를 하나 할당해서 S_p를 0으로 해서 하위 반복 패턴이 존재함을 표시하고, 최소 반복 단위를 구성하는 PTE를 구성하는 PTE개수를 I_p에 기록하고 최소 반복 단위를 몇 번 반복하는지 반복 회수를 C_p에 기록한다. 바로 이어서 최소 반복 단위를 이루는 PTE를 기록함으로써 복합적인 PTE의 반복을 압축해서 표현할 수 있다.

그림 7의 PTE는 최소 반복 단위가 2개의 PTE로 이루어지므로 I_p는 2이고 이 반복단위가 N번 반복되므로 C_p는 N으로 기록하고 바로 이어서 최소 반복단위를 이루는 2개의 PTE를 기록하여 PTE를 압축하는 것을 보여주고 있다.

4.3 파일 액세스 패턴 정보의 저장 및 관리

하나의 파일에 액세스하는 응용프로그램은 하나 혹은 여러 개일 수 있다. 또한 하나의 응용프로그램이라 하더라도 실행 옵션 등에 의해 여러 가지 액세스 패턴을 가질 수 있다. 한 파일을 액세스하는 패턴이 여러 개 존재할 경우, 이러한 각각의 패턴을 경로라고 한다.

SIC 파일 액세스 패턴 수집에서는 각 경로를 하나의 데이타로 하여 독립적으로 관리한다. 즉 경로의 개수만큼의 액세스 패턴 정보가 존재하게 된다. 경로의 개수가 많이 기록되어 있으면 미래에 나타날 액세스 패턴이 현재 기록되어 있는 경로 중에 존재할 가능성이 높아지게 된다. 하지만, 너무 많은 경로가 존재하면 각 액세스 패턴 정보가 차지하는 저장공간이 많이 필요하게 되고 선반입 수행에 사용하는 것도 어려워진다.

따라서, 본 논문에서는 이러한 각 경로들을 저장할 수 있는 최대 개수를 지정하고 LFU(Least Frequently Used) 교체정책으로 관리하도록 하였다. LFU 교체정책은 기록된 경로의 개수가 최대 허용 경로 개수가 되면 새로운 액세스 패턴 정보를 기록하기 위해 현재 기록된 경로 가운데 가장 발생 회수가 작은 경로를 제거하고 새로운 경로를 기록하는 것이다. 이렇게 함으로써 발생 빈도가 높은 경로를 계속 유지할 수 있게 된다.

5. SIC 기법에서의 선반입 수행

파일 액세스 패턴 수집에서 수집된 액세스 패턴은 파일이 다음에 열리면 선반입 수행 부분에서 이를 사용해서 선반입을 수행하게 된다.

여기에서는 선반입 정책을 수행하기 위해 결정되어야 하는 인자들을 고찰하고 SIC 액세스 패턴 정보를 사용해서 이러한 인자를 결정하는 방법을 설명한다.

5.1 선반입의 대상

선반입의 대상을 결정하는 것은 바로 다음에 사용될

데이타 블록을 예측하는 것이다.

SIC 선반입 기법에서는 과거에 나타난 액세스 패턴을 그대로 나타낼 가능성이 높다고 가정하고 현재의 액세스와 과거에 나타난 액세스 패턴을 비교하여 동일하다면 이후에 나타날 액세스는 액세스 패턴 정보에 기록된 것과 같을 것으로 가정한다. 따라서, SIC 선반입 기법에서 선반입의 대상은 현재의 액세스와 일치하는 액세스 패턴 정보에 나타난 액세스의 바로 다음 액세스로 결정된다.

그림 8은 SIC 선반입 기법에서 선반입할 데이타 블록을 결정하는 방법을 보여준다.

그림 8에서 보인 예는 현재 기록하고 있는 액세스 패턴 정보의 S_p와 I_p가 이전에 기록된 액세스 패턴 정보의 S_p와 I_p와 각각 일치하므로 동일한 액세스 패턴임을 알 수 있다. 현재 기록되고 있는 액세스 패턴 정보의 C_p가 선반입에 사용하고 있는 액세스 패턴 정보의 C_p보다 작으므로 다음에도 이 패턴의 읽기 크기와 읽기 요청간의 간격이 나타나게 될 것으로 예상된다. 따라서, 현재 액세스하고 있는 데이타 블록에서 I_0 떨어진 곳에서 S_0만큼을 액세스할 것으로 예상하고 선반입을 수행하게 된다.

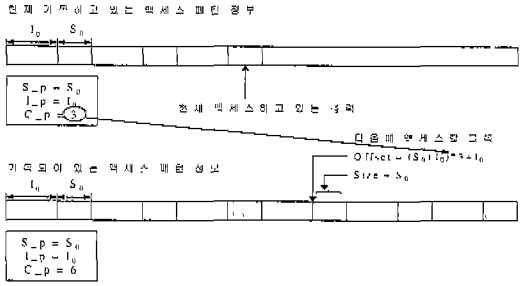

그림 8 SIC 액세스 패턴 정보를 사용해서 선반입 대상을 결정하는 방법

5.2 선반입의 양

선반입의 양은 한번에 선반입할 데이타의 양을 뜻한다. 너무 많은 양을 선반입하면 캐시 버퍼내의 많은 데이타 블록을 교체하게 되어 캐시 적중률을 저하시킬 확률이 높아지고 너무 적은 양을 선반입하면 디스크 입출력 회수가 많아져서 오버헤드가 커지게 된다.

디스크 입출력에 걸리는 시간은 읽어들이는 크기와 읽기 요청의 회수에 의해 결정된다. 즉, 같은 크기라도 읽기 요청 회수가 많으면 더 많은 시간이 걸리고, 같은 읽기 요청회수라도 크기가 크면 더 많은 시간이 걸리는 것이다. SIC 선반입 기법에서 선반입하는 최소 단위는 응용프로그램의 액세스 한번과 같다. 따라서, SIC 선반입 기법에서의 선반입양은 액세스의 회수로 나타내어진다.

선반입 양이 최적량보다 크면 캐시 적중률은 그대로지만 선반입에 걸리는 시간이 늘어나서 응답속도가 저하된다. 반대로 선반입 양이 최적량보다 작으면 캐시 적중률이 저하되어서 추가 입출력이 늘어나게 되어 응답속도가 저하된다. 따라서, 최적의 선반입 양은 캐시 적중률과 파일 입출력 속도가 최대이면서 최소의 선반입 양이 되는 지점에서 결정된다.

5.3 선반입의 시기

최적의 선반입 시기는 응용프로그램이 데이타 블록을 요청했을 때는 캐시 버퍼에 들어가 있을 정도로 충분히 빠른 시간이지만, 선반입된 이후에 응용프로그램이 요청할 때까지 걸리는 시간이 가장 짧은 시기이다. 따라서, 본 논문에서는 최적의 선반입 시기를 캐시 적중률을 저하시키지 않는 한도 내에서 가장 늦은 시간으로 결정하였다.

5.4 선반입 과정

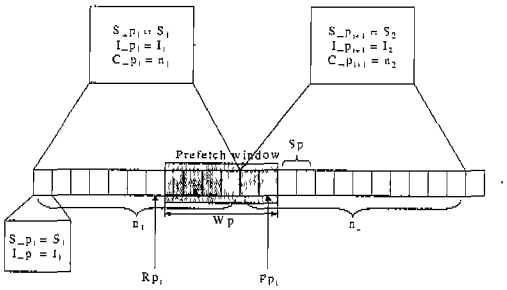

그림 9 SIC 선반입 수행의 인자

그림 9는 SIC 기법의 각 인자를 나타내고 있다. 가운데의 음영부분 Wp가 선반입 윈도우의 크기를 나타낸다. 줄지어 서있는 사각형들은 SIC 액세스 패턴 정보의 각 액세스들을 압축 해제해서 하나하나 늘어놓은 것이다. 각 사각형은 읽기 크기와 읽기 요청간의 거리를 가지고 있다.

Rp_t는 가장 최근에 응용프로그램이 요청한 읽기 액세스를 가리키고 있다. Pp_t는 가장 최근에 선반입이 수행된 액세스를 가리킨다. 선반입 윈도우는 Rp_t와 함께 이동한다. 이 때, 선반입 윈도우의 크기가 크면 현재 응용프로그램의 읽기 요청에 비해 더 먼 미래에 요청할 액세스까지 선반입을 수행하게 되고 선반입 윈도우의 크기가 작으면 좀더 가까운 미래에 요청할 액세스까지 선반입을 수행하게 된다. 즉, 선반입 윈도우의 크기에 따

라 현재 응용프로그램의 읽기보다 얼마나 더 미래의 읽기 요청까지 선반입할 것인지가 결정되는 것이다.

최적의 선반입 윈도우의 크기는 시스템의 파일 입출력 성능과 캐시 버퍼의 크기 등에 의해 결정된다. 윈도우의 크기가 최적의 크기보다 작으면 응용프로그램이 요청하기 전에 캐시 버퍼에 데이타 블록이 저장되지 않아서 캐시 적중률을 저하시키고 응용프로그램의 파일 입출력의 응답시간이 길어진다. 반대로 최적의 크기보다 크다면 캐시 버퍼에 너무 많은 선반입이 되고 이후 사용되지 않은 데이타 블록이 교체되어 캐시 버퍼의 사용 효율이 떨어지게 된다.

5.5 액세스 패턴 정보 선택

앞에서 설명한 선반입 방법은 현재의 액세스와 일치하는 액세스 패턴 정보를 알고 있을 때에 해당하는 것이다. 파일 액세스 패턴 수집 부분에서 언급한 바와 같이 여러 개의 경로를 가지는 파일은 액세스 패턴 정보 또한 여러 개가 존재하게 된다.

선반입은 응용프로그램이 미래에 요청할 것으로 예측되는 데이타 블록을 미리 읽어들이는 것이다. 따라서, 선반입이 수행되는 시점에서는 여러 개의 경로 중에서 응용프로그램의 액세스가 어떤 경로와 일치하는지를 알 수 없다. 따라서, 가장 확률이 높은 경로를 선택할 수 있는 방법이 필요하다.

파일 액세스 패턴 수집 부분에서 하나의 파일에 대해 여러 개의 경로가 존재하면 각 경로에 해당하는 액세스 패턴 정보를 LFU 교체 정책으로 관리하는 것에 대해 언급하였다. LFU 교체 정책은 각 경로의 발생 회수를 기록해두고 가장 발생회수가 작은 경로를 교체하는 것이다. 따라서 각 경로에는 발생회수 정보가 포함되어 있다.

SIC 선반입 기법에서는 선택대상이 되는 액세스 패턴 정보가 여러 개라면 그 중에서 가장 발생회수가 높은 액세스 패턴 정보가 현재 응용프로그램의 액세스와 일치할 가능성이 높다고 보고 가장 발생회수가 많은 액세스 패턴 정보를 사용해서 선반입을 수행한다.

그림 10은 액세스 패턴 정보를 선택하는 방법을 보여주고 있다.

파일이 처음 열리면 파일에 해당하는 모든 액세스 패턴 정보들을 목록에 추가한다. 이후에 응용프로그램의 읽기 요청이 있으면 읽기 요청의 인자와 액세스 패턴 정보목록을 비교하여 일치하지 않는 액세스 패턴 정보를 목록에서 제거한다. 그리고 나서, 남은 액세스 패턴 정보 중에서 가장 발생 회수가 높은 액세스 패턴 정보를 사용해서 선반입을 수행한다.

이와 같은 과정을 응용프로그램의 읽기 요청이 발생

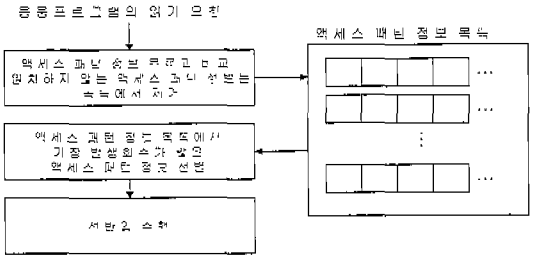

그림 10 액세스 패턴 선별 방법

할 때마다 수행하게 된다. 만약, 모든 액세스 패턴 정보가 목록에서 제거되었다면, 이것은 현재의 액세스와 일치하는 액세스 패턴 정보가 없다는 뜻이므로 이때는 SIC 선반입을 수행할 수 없다.

6. 실험 및 성능 평가

이 장에서는 리눅스 VFS 상에 구현된 SIC 선반입 시스템의 성능을 실험한 결과를 제시한다.

6.1 실험 환경

SIC 선반입 시스템은 리눅스 운영체제의 커널 2.2.17을 수정하여 구현하였다.

사용된 시스템의 사양은 다음과 같다.

CPU 펜티엄III 1GHz

RAM 256MB

Storage 40GB EIDE Hard disk

성능 실험을 위해 응용프로그램의 액세스 패턴을 다음과 같이 선정하였다.

패턴 1 : 순차적, 읽기, 100MB

패턴 2 : 단순 스트라이드, 읽기, 100MB,

패턴 3 : 랜덤, 읽기, 100MB, 1-50KB read, 랜덤 lseek

패턴 1은 일반적인 시스템의 파일 액세스의 대부분을 차지하는 순차적 읽기를 사용하여 일반적인 액세스 패턴에서의 성능을 비교하기 위한 것이다. 패턴 2는 병렬처리 과학 기술 계산 응용에서 자주 나타나는 패턴으로 일반적인 OBL나 IBL에서는 선반입을 수행할 수 없는 패턴으로서 새로운 선반입 정책이 어느 정도의 성능 향상을 보이는 지를 알아보기 위한 것이다. 패턴 3은 패턴을 기록할 수 없는 패턴을 사용하여 제안된 선반입 정책이 이러한 패턴에 대해서는 성능이 좋지 않음을 보이기 위한 것이다.

6.2 SIC 선반입 인자

최적의 성능을 내는 선반입 양과 선반입 윈도우의 크기는 파일 입출력 시스템의 성능과 캐시 버퍼의 크기 등의 실제 시스템의 성능에 의해 결정된다.

본 논문에서는 실험적으로 최적의 성능이 얻어지는 선반입 양을 구했다. 실험에 의해 결정된 최적의 선반입 양은 읽기 크기(size)가 1 페이지(=4KB) 이하일 때는 2 페이지(=8KB) 이고, 읽기 크기가 1 페이지보다 클 때는 읽기 크기의 2배이다. 이보다 적은 양을 선반입하게 되면 캐시 적중률이 저하되게 되고, 이보다 많이 선반입하게 되면 캐시 적중률은 거의 같지만, 처음 액세스가 시작될 때, 선반입에 걸리는 시간이 길어져서 첫 번째 읽기 요청에 대한 응답시간이 길어진다.

$$\text{최적의 선반입 양} = \begin{cases} = 2\ \text{pages}\ (=8\text{KB})\ \text{if}\ S\_p < 1\text{page}\ (=4\text{KB}) \\ = 2 \times S\_p\ \text{if}\ S\_p >= 1\text{page}\ (=4\text{KB}) \end{cases}$$

본 논문에서 실험적으로 알아낸 최적의 선반입 시기가 되는 선반입 윈도우의 크기는 선반입 양의 3배이다.

최적의 선반입 윈도우의 크기 = 3 × 선반입 양

이보다 더 작은 크기에서는 캐시 적중률이 저하되고 응용프로그램의 수행속도가 떨어졌고, 이보다 더 큰 크기에서는 캐시 적중률은 거의 일정하지만 응용프로그램의 첫 번째 읽기 요청 시에 선반입하는 양이 많아져서 응답속도가 떨어졌다.

6.3 실험 결과

SIC 선반입 기법은 이전에 저장된 액세스 패턴 정보와 동일한 액세스 패턴에 대해서만 선반입을 수행할 수 있다. 실험에서는 각 액세스 패턴에 대해 한 번의 실행을 거쳐서 액세스 패턴 정보를 수집할 수 있도록 하고 실험을 실시하였다.

선정한 각 액세스 패턴에 대해 5회의 측정을 실시하였고 이 중에서 최대와 최소를 제거한 나머지 3회의 측정값을 평균해서 결과를 도출하였다. 비교대상이 되는 정책은 리눅스의 OBA 정책으로서 OBL 정책의 변형이다. 리눅스 OBA 정책은 순차적 읽기에 최적화되어 있고 순차적 읽기가 계속되면 선반입하는 크기를 늘리며, 정해진 크기 이상의 lseek가 발생하면 선반입을 중지한다. 이후에 순차적 읽기가 감지되면 다시 선반입하는 크기를 조금씩 늘려간다.

6.3.1 순차적 액세스 패턴

순차적 읽기 액세스 패턴에 대해 리눅스의 선반입과 구현된 선반입 정책을 적용하여 실험을 실시하였다. 테스트 프로그램은 한번에 1KB-64KB 크기를 파일의 처음부터 끝까지 순차적으로 읽었다.

표 1은 순차적 읽기 액세스 패턴에 대한 수행 시간을 읽어들이는 단위를 1KB에서 64KB까지 변화시키면서 측정한 결과이다. 그림 11은 표 1의 결과를 그래프로 나타낸 것이다.

표 2는 이 실험에서 측정된 캐시 적중률로서 선반입의 정확성을 보여주고 있다.

표 1 순차적 읽기 수행 시간 결과 (단위: 초)

	1KB	2KB	4KB	8KB	16KB	32KB	64KB
Linux OBA	44.39	21.80	11.09	10.42	9.02	9.06	9.47
Hint based	41.37	21.80	10.33	7.34	7.35	7.27	7.09
SIC scheme	44.39	21.81	10.63	7.52	7.57	7.57	7.58

표 2 순차적 읽기 수행 캐시 적중률 (단위 : %)

	1KB	2KB	4KB	8KB	16KB	32KB	64KB
Linux OBA	98.74	97.90	96.86	96.86	96.87	96.88	96.88
Hint based	99.99	99.99	99.99	99.99	99.99	99.99	99.99
SIC scheme	99.99	99.99	99.99	99.98	99.98	99.96	99.99

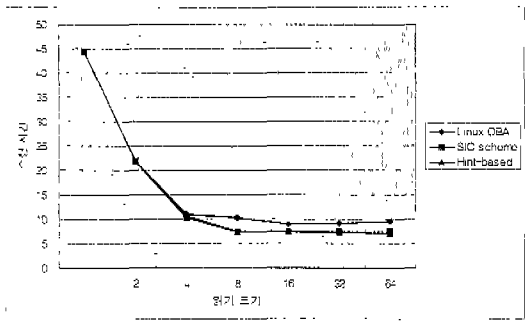

그림 11 순차적 읽기

그림 11에서 볼 수 있듯이 4KB 이하의 읽기 크기에서는 거의 같은 성능을 내지만 그 이상의 크기에 대해서는 최소 16%에서 최대 27%의 성능 향상이 있음을 확인할 수 있다.

또한, 수행 시간이 1KB에서 4KB 까지는 급격히 감소하다가 4KB 이후에는 거의 일정한 것을 볼 수 있는데, 이것은 리눅스에서 사용하는 최소 입출력 단위가 4KB이기 때문이다. 즉, 1KB를 4번 연속 접근하는 것은 4KB 크기를 4번 접근하는 것과 같기 때문에 4배정도의 시간이 걸리게 된다. 하지만 4KB 크기를 넘는 액세스의 경우에는 이러한 중복 접근이 발생하지 않기 때문에 읽어들이는 크기가 같다면 거의 같은 시간이 걸리게 된다.

표 2에 나타난 바와 같이 리눅스의 선반입 정책은 순

차적 읽기에 최적화되어 있어서 96%이상의 캐시 적중률을 유지하는 것을 알 수 있다. 또한, 제안한 SIC 정책에서는 순차적인 액세스 패턴에 대해 거의 100%에 가까운 확률로 캐시 적중률을 향상시킬 수 있음을 확인할 수 있다.

그리고 힌트 기반 선반입 기법과의 결과를 보면 거의 성능차이가 없거나 힌트기반의 선반입 기법이 근소한 성능 개선이 있음을 알 수 있다. 이는 힌트 기반의 선반입 기법은 응용프로그램 자체에서 자신의 액세스 패턴을 미리 파일 시스템에게 알려주므로 어떤 면에서 더 뛰어난 성능을 발휘한다고 할 수 있다. 하지만 힌트기반의 선반입 기법은 응용프로그램에서 먼저 액세스 패턴을 파일 시스템에게 알려주어야 하는 부가적인 작업이 있으며, 또한 본 논문에서 실험한 액세스 패턴은 한번 액세스한 패턴은 다음 액세스시에 같은 패턴으로 액세스한다고 가정하고 있어 성능면에서는 거의 차이가 없다. 그리고 힌트 기반의 선반입 기법은 응용프로그램 자체에서 자신의 액세스 패턴을 알려주어야 하는 루틴을 새로 작성하여야 하며, 또한 파일시스템 자체에서도 이 결과를 받아들여야 하는 루틴을 만들어야 하는 개발면에서의 과부하가 SIC 선반입 기법에 비해 성능면이 떨어진다고 할 수 있다.

6.3.2 단순 스트라이드 액세스 패턴

단순 스트라이드 패턴은 과학 기술 계산을 위한 병렬 응용프로그램에서 발견되는 액세스 패턴으로서 일정한 길이의 액세스와 일정한 길이의 lseek을 반복하는 것이다. 여기에서는 lseek 거리를 고정하고 읽기의 크기를 변화시켜서 그 결과를 관찰하였다.

표 3은 lseek 거리를 1024byte로 고정하고 읽기 크기를 1KB에서 64KB까지 변화시키면서 그 읽기에 걸린 수행시간을 기록한 것이다. 그림 12는 수행시간 결과를 그래프로 나타낸 것이다. 표 4는 캐시 적중률을 보여주고 있다.

SIC 정책은 리눅스의 OBA에 비해 1024byte의 lseek 거리를 가지는 단순 스트라이드 패턴에서 1KB의 읽기 크기에서는 3.4% 정도의 성능저하가 발생하였지만 그보다 큰 읽기 크기에서는 최대 25%정도의 성능 향상을 보이고 있다. 하지만, lseek거리가 짧기 때문에 읽기 크기가 커지면서 점점 순차적 읽기와 거의 동일한 성능과 캐시 적중률을 보이고 있다.

표 3 1024 byte 단순 스트라이드 패턴에서의 읽기 수행 시간(단위: 초)

단위: 초	1KB	2KB	4KB	8KB	16KB	32KB	64KB
Linux OBA	21.87	14.89	11.06	10.08	9.16	8.90	9.39
Hint based	21.74	13.90	8.20	8.20	7.75	7.92	7.84
SIC scheme	22.63	14.36	8.28	8.11	7.55	7.93	7.83

표 4 1024 byte 단순 스트라이드 패턴에서의 캐시 적중률(단위: %)

단위: %	1KB	2KB	4KB	8KB	16KB	32KB	64KB
Linux OBA	97.64	97.53	97.29	97.07	97.01	96.94	96.91
Hint based	99.99	99.99	99.99	99.99	99.99	99.99	99.99
SIC scheme	99.99	99.99	99.83	99.80	99.98	99.87	99.90

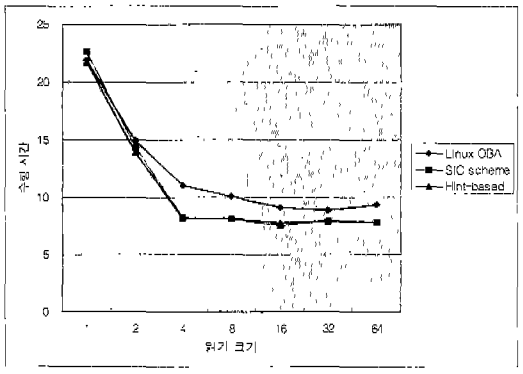

그림 12 단순 스트라이드 읽기

6.3.2 랜덤 액세스 패턴

선반입 패턴 수집기는 1000개의 제한된 길이보다 긴 패턴이 발생하면 패턴 수집을 중지하게 된다. 또한 구현된 선반입 정책은 이전의 액세스 패턴이 기록되어 있지 않은 상황에서는 리눅스의 선반입을 그대로 사용하도록 하였다. 따라서 랜덤한 액세스 패턴에 대해서는 리눅스의 선반입 정책과 같은 성능을 보이게 된다.

표 5는 랜덤한 액세스 패턴에 대해 실험을 수행하였을 때의 수행 시간을 기록한 것이다.

표 5 랜덤 패턴에서의 읽기 수행 시간(단위: 초)

	1회	2회	3회	4회	5회	평균
Linux OBA	37.22	31.24	29.54	40.12	35.12	34.65
SIC scheme	35.21	35.24	37.54	34.23	37.12	32.54

실험에 사용된 패턴은 1-64KB의 읽기와 0-16KB의 lseek이 발생하도록 하였고 읽기 크기가 107MB가 되는

데 까지 걸리는 시간을 측정하였다.

결과에서 보듯이 두 정책은 거의 동일한 성능을 보인다. 리눅스의 OBA 선반입 정책은 순차적 읽기에 최적화되어 있기 때문에 랜덤한 읽기 패턴에 대해서는 선반입을 수행하지 않는다. 또한 SIC 정책은 패턴의 길이가 1000개 보다 길면 패턴 수집을 하지 않기 때문에 역시 선반입을 수행하지 않는다.

6.4 실험 결과 고찰

SIC 선반입 시스템은 과거에 나타난 액세스 패턴과 동일한 액세스 패턴이 발생하는 경우에 선반입을 수행할 수 있다. 이전에 발생한 액세스 패턴이 똑같이 재현되는 상황에서는 SIC 선반입 시스템은 리눅스의 OBA 선반입 기법에 비해 평균적으로 20% 이상의 성능 향상을 이룰 수 있음을 알 수 있다.

최고의 성능 차이를 보이는 것은 읽기 크기가 4KB-8KB일 때로서 최대 40%정도의 성능 개선을 보인다. 하지만 액세스 크기가 4KB 보다 작고 순차적인 액세스에 대해서는 리눅스의 OBA와 거의 같은 성능을 보인다. 또한 한번에 읽는 크기가 16KB 이상으로 커지면 성능의 차이가 줄어들고 있다.

랜덤한 액세스 패턴에 대해서는 액세스 패턴을 기록할 수 없기 때문에 SIC 선반입 기법을 적용할 수 없다. 이에 따라 SIC 선반입 시스템에서도 OBA 선반입 기법으로 선반입을 수행하기 때문에 거의 동일한 성능이 측정되었다.

위의 결과는 지식기반 선반입 기법이 일반적인 OBA 선반입 기법에 비해 성능이 뛰어남을 보여준다. 여기에 더해 SIC 선반입 시스템은 액세스 패턴 정보 저장에 필요한 메모리 공간을 기존의 지식기반 선반입 기법에 비해 크게 줄이고 있다. 위의 실험에 사용된 액세스 패턴의 경우에는 최대 50 byte 정도의 메모리 공간만을 사용함으로써 기존의 지식 기반 선반입 기법이 최소 수 KB에서 최대 수십 KB의 메모리 공간을 사용하는 것에 비해 메모리 사용 효율이 매우 높음을 알 수 있다.

7. 결론 및 향후 과제

현재의 파일 액세스 패턴은 멀티미디어 파일의 증가와 네트워크 대역폭의 확대에 따라 대용량의 파일이 증가하고 재사용의 빈도는 감소하고 있다. 이러한 상황에서는 캐시 적중률이 감소하게 된다. 캐시 적중률을 향상시키기 위해서 선반입의 필요성이 높아지고 있지만, 잘못된 선반입은 파일 입출력의 성능을 저하시키게 된다.

따라서 정확한 선반입을 통해 파일 입출력의 성능을 향상시키기 위한 목적으로 응용프로그램의 액세스 패턴에 따라 동적으로 선반입을 수행하는 지식 기반 선반입 방법이 고안되었다. 하지만, 기존의 지식 기반 선반입 방법은 응용프로그램의 액세스 패턴 정보를 저장하는데 필요한 메모리 공간이 파일의 길이에 비례해서 커지는 문제점이 있다.

본 논문에서 제안하는 SIC 선반입 기법은 이전에 수행된 파일 액세스 패턴에 관한 연구에 의해 알려진 파일 액세스 패턴의 반복성에 기반 하여 액세스 패턴 정보를 압축해서 저장함으로써 지식 기반 선반입 방법의 성능을 그대로 유지하면서 메모리 사용량을 줄였고 수집된 액세스 패턴 정보를 디스크에 파일로 저장함으로써 지식 기반 선반입 방법의 문제점인 시스템 리셋 등에 의해 수집된 액세스 패턴 정보가 소실되는 문제를 개선하였다. 또한, 액세스 패턴 정보를 메모리에 캐시해둠으로써 파일 개방 시에 선반입 수행을 위해 액세스 패턴 정보를 읽어들이는 오버헤드를 경감시켰다.

실험 결과에서 알 수 있듯이, 현재 가장 많이 사용되고 있는 OBA 선반입 기법이 순차적 액세스 패턴에 대해서만 충분한 성능을 보여주는데 비해서 제안하는 SIC 선반입 기법은 순차적 액세스를 포함해서 단순 스트라이드 액세스 패턴에 대해서도 선반입을 수행해서 성능을 향상시킬 수 있다.

현재 SIC 선반입 기법은 과거에 나타난 액세스 패턴과 동일한 액세스에 대해서만 선반입을 수행할 수 있다. 향후 과제로는 과거의 액세스 패턴과 완전히 동일하지는 않지만 거의 유사한 액세스에 대해서도 선반입을 적용할 수 있는 방법에 대한 연구가 필요하다.

참 고 문 헌

[1] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," In Proc. of IEEE Infocom '99, pp. 126-134, March 1999.

[2] T. M. Madhyastha, "Automatic Classification of Input/Output Access Patterns, Tech. Rep.," University of Illinois at Urbana-Champaign, Department of Computer Science, August 1997.

[3] J. K. Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson, "A Trace-Driven Analysis of the UNIX 4.2 BSD File System," In Proc. of the 10th Symposium on Operating System Principles, pp. 15-24, December 1985.

[4] N. Nieuwejaar, D. Kotz, A. Purakayastha, C.S. Ellis, and M. Best, "File-access characteristics of parallel scientific workloads," IEEE Transactions

on Parallel and Distributed Systems, 7(10):1075-1089, October 1996.

[5] 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, pages 165-172, April, 1995.

[6] D. Kotz and C.S. Ellis. "Practical prefetching techniques for multiprocessor file systems," Journal of Distributed and Parallel Databases, 1(1):33-51, January 1993.

[7] T. M. Madhyastha and D. A. Reed, "Exploiting Global Input/Output Access Pattern Classification," In Proc. of SC'97, November 1997, CD-ROM.

[8] T. M. Madhyastha and D. A. Reed, "Input/Output Access Pattern Classification Using Hidden Markov Models," In Proc. of the Workshop on Input/Output in Parallel and Distributed Systems (IOPADS), November 1997.

[9] R.H. Patterson, G.A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. "Informed prefetching and caching." In Proc. of the Fifteenth ACM Symposium on Operating Systems Principles, pp. 79-95, December 1995.

[10] A. Tomkins, R.H. Patterson and G.A. Gibson, "Informed Multi-Process Prefetching and Caching," In Proc. of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), June 1997.

황보 준 형

2000년 경북대학교 전자전기공학부 졸업 (학사). 2000년 ~ 현재 경북대학교 전자공학과 (석사과정). 관심분야는 병렬처리, 분산처리, 클러스터 컴퓨팅, 암호화 시스템

석 성 우

1999년 경북대학교 전자전기공학부 졸업 (학사). 2001년 경북대학교 전자공학과 졸업(석사). 2001년 ~ 현재 한국전자통신연구원 컴퓨터 소프트웨어 연구소 연구원. 관심분야는 SAN기반 스토리지 시스템, 하드웨어 디바이스드라이버 개발, 로우레벨 프로그래밍

서 대 화

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. 관심분야는 병렬분산처리, 운영체제, 병렬처리, 컴퓨터 구조