PMS : 다단계 저장장치를 고려한 효율적인 선반입 정책

(PMS : Prefetching Strategy for Multi-level Storage System)

이 규 형 † 이 효 정 † 노 삼 혁 ††
(Kyu Hyung Lee) (Hyo Jeong Lee) (Sam Hyuk Noh)

요 약 저장장치를 다단계로 구성하여 효율적으로 많은 사용자의 요청을 동시에 처리하는 다단계 저장장치의 활용은 점차 늘어나고 있다. 저장 장치가 다단계로 발전하여, 매우 많은 데이타를 효과적으로 처리할 수 있게 되었으나, 디스크에 접근하기 위한 단계가 늘어남으로써 성능이 저하되는 문제가 발생한다. 현재까지는 메모리와 프로세스에 비해 느린 디스크 접근 속도를 완충하기 위해 선반입 정책이 매우 효과적이었다. 그러나 기존의 선반입 기법은 대부분 다단계로 구성된 저장장치는 고려하지 않기 때문에 다단계 저장장치에서 기존의 선반입 기법을 사용할 경우 큰 성능향상을 기대 할 수 없다. 본 연구에서는 네트워크로 연결된 다단계 저장장치에서 상위 레벨의 선반입 기법에 의존하지 않는, 넓은 용도로 사용할 수 있는, Prefetching Strategy for Multi-level Storage system(PMS)라 칭하는 하위 레벨 선반입 기법을 제안하였다. 이는 시스템의 사용자, 어플리케이션 혹은 상위 시스템과 독립적으로 동작하기 때문에 단지 하위 시스템의 선반입 정책으로 적용함으로써 쉽게 높은 성능을 사용할 수 있다. 또한 PMS 정책의 성능을 측정하기 위해 본 연구에서는 실제 시스템을 정교하게 흉내 내는 시뮬레이터를 개발하여 널리 쓰이는 두 가지 트레이스를 이용한 서로 다른 32가지의 실험을 하였고, 기존의 선반입 정책을 하위 레벨에 적용한 시스템에 비해 PMS 정책을 하위 레벨에 적용할 경우, 모든 경우에서 성능향상을 확인 할 수 있었고, 최대 35%, 평균 16.56%의 평균 응답시간이 좋아짐을 보였다.

키워드 : 선반입, 다단계 저장장치

Abstract The multi-level storage architecture has been widely adopted in servers and data centers. However, while prefetching has been shown as a crucial technique to exploit sequentiality in accesses common for such systems and hide the increasing relative cost of disk I/O, existing multi-level storage studies have focused mostly on cache replacement strategies. In this paper, we show that prefetching algorithms designed for single-level systems may have their limitations magnified when applied to multi-level systems. Overly conservative prefetching will not be able to effectively use the lower-level cache space, while overly aggressive prefetching will be compounded across levels and generate large amounts of wasted prefetch. We design and implement a hierarchy-aware lower-level prefetching strategy called PMS(Prefetching strategy for Multi-level Storage system) that applicable to any upper level prefetching algorithms. PMS does not require any application hints, a priori knowledge from the application or modification to the I/O interface. Instead, it monitors the upper-level access patterns as well as the lower-level cache status, and dynamically adjusts the aggressiveness of the lower-level prefetching activities. We evaluated the PMS through

· 이 논문은 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업으로 수행된 연구임(No. R0A-2007-000-20071-0)
· 이 논문은 2008 한국컴퓨터종합학술대회에서 'PMS : 다단계 저장장치를 고려한 효율적인 선반입 정책'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 홍익대학교 컴퓨터공학과
lkh@cs.hongik.ac.kr
hjlee@mail.hongik.ac.kr
†† 종신회원 : 홍익대학교 컴퓨터공학과 교수
samhnoh@hongik.ac.kr

논문접수 : 2008년 8월 25일
심사완료 : 2008년 11월 3일

정보과학회논문지: 시스템 및 이론 제36권 제1호(2009.2)

extensive simulation studies using a verified multi-level storage simulator, an accurate disk simulator, and access traces with different access patterns. Our results indicate that PMS dynamically controls aggressiveness of lower-level prefetching in reaction to multiple system and workload parameters, improving the overall system performance in all 32 test cases. Working with four well-known existing prefetching algorithms adopted in real systems, PMS obtains an improvement of up to 35% for the average request response time, with an average improvement of 16.56% over all cases.

Key words : Prefetching, Multi-level storage system

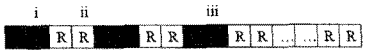
1. 서 론

최근의 저장 장치는 많은 사용자들이 동시에 요청하는 대용량의 자료를 처리하기 위해 다단계로 확장하는 추세이다. 하지만 CPU와 메모리의 발달로 인해 데이타 I/O는 전체 시스템의 성능을 해치는 병목구간이며, 다단계 저장장치의 경우 데이타에 접근하기 위한 과정이 늘어남으로 전체 시스템의 성능을 더욱 떨어뜨릴 수 있다. 이러한 저장장치의 약점을 극복하기 위해 이미 요청된 데이타를 버퍼 캐쉬에 저장해 놓음으로써 성능 향상을 꾀하는 방법과 필요한 데이타를 미리 가져다 놓음으로 성능을 향상시키는 선반입 기법이 오래 전부터 연구되어 왔다. 다단계 저장장치를 고려한 버퍼 캐쉬 교체 정책은 여러 가지로 연구가 되어 왔으나[1-3], 다단계 저장 장치를 고려한 선반입 기법에 대한 연구는 현재까지 거의 이루어 지지 않았다. 하지만, 선반입 기법을 잘 활용한다면, 다단계 저장장치의 각 단계에서 걸리는 I/O 시간을 효과적으로 감출 수 있기 때문에 다단계 저장장치에서 선반입 기법은 성능향상을 위해 대단히 중요한 역할을 할 수 있을 것으로 기대된다.

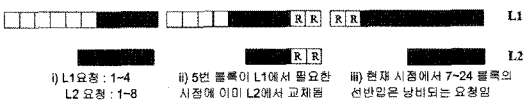
선반입 기법은 향후 필요할 것으로 예상되는 데이타를 미리 디스크에서 읽어 오는 방법이다. 일반적으로 현재 필요한 데이타를 디스크에서 읽어올 때 추가로 일정량을 미리 읽어 옴으로써 향후 연속적인 데이타 요청에 대비를 하는 방법을 사용한다. 그러나, 앞으로의 요청을 정확하게 예측하지 못할 경우엔 필요하지 않을 데이타를 미리 가져옴으로써 캐쉬공간과 디스크 사용의 낭비를 초래하거나 미리 데이타를 준비하지 못함으로써 선반입으로 인한 이득을 누릴 수 없게 된다. 과거의 1단계 시스템이라면, 부정확한 예측으로 인한 선반입의 부작용이 크지 않기 때문에 과감한 선반입 정책을 사용할 수 있었다. 그러나, 다단계 시스템의 각 단계에서 개별적인 선반입을 수행한다면, 부정확한 선반입으로 인한 낭비가 1단계 시스템일 때와는 비교할 수 없이 크게 늘어나게 된다.

그림 1은 2단계로 구성된 저장장치이고, 각 단계에서는 개별적인 선반입 기법이 적용되었다. 각 시스템은 네트워크로 연결되어 있으며, 디스크는 하위 시스템에 직접 연결된 형식이다. 그림 2의 (a)와 (b)에서는 부정확

그림 1 다단계 저장장치 구조



(a) 어플리케이션의 엑세스 패턴. 짙은 블록은 순차적 접근을 나타내며 밝은 블록은 무작위 접근을 나타냄



(b) 시간 순서에 따른 버퍼 캐쉬의 상태. (a)의 i, ii, iii의 상황에 따라 L1과 L2 캐쉬의 블록 저장 상태

그림 2 부정확한 선반입 예측으로 인한 낭비 예제

한 선반입 예측이 두 단계의 시스템을 거치면서 발생시키는 부작용을 예를 들어 설명하였다. (a)는 어플리케이션에서의 블록 요청 순서를 나타낸다. 숫자가 있는 짙은 색 블록은 선반입 정책으로 이득을 볼 수 있는 순차적인 접근 패턴을 나타내고, 밝은 색 블록은 선반입이 아무런 도움이 되지 않는 무작위적인 접근을 뜻한다. 이 예제에서는 상위 레벨(L1)과 하위 레벨(L2) 시스템 모두 독립적인 선반입 정책을 사용하고 있는 상황이고, 상위와 하위 레벨의 캐쉬 크기는 각각 10블록과 5블록으로 구성되어 있다. 이때 i, ii, iii의 세 가지 시점에서 L1과 L2의 버퍼캐쉬의 상황을 (b)에서 자세히 살펴 본다. 먼저, 2번 블록이 요청되는 시점인 (i)에서 상위 레벨 시스템(L1)은 3과 4의 블록을 선반입 요청한다. 이 요청을 받은 하위 레벨 시스템(L2)은 자체적인 선반입 기법으로 2~8번까지의 블록을 요청하지만, 버퍼 캐쉬의 크기가 5블록이기 때문에 실제 4~8번 블록이 저장된다. (ii)의 시점에서는 무작위 블록이 요청되었고, 이 요청으로 인해 L1에서는 요청된 블록 외에 연속되는 한 블록이 추가로 선반입 요청이 된다. 이후, 어플리케이션이 5번 블록을 필요로 하는 시점에서 L1은 아직 5번을 선반입 하지 않은 상황이고, L2는 5번 블록을 (i)에서 선반입을 했으나, 캐쉬 크기의 한계로 인해 이미 교체되어

사라진 이후이다. 즉, 선반입이 너무 일찍 일어나 아무런 이득을 보지 못하고 오히려 캐쉬 공간의 낭비와 디스크 I/O의 손해만을 일으켰다. 또한 (iii)의 상황에서는 L1에서는 요청된 6번 블록에 이어 7~12번 블록까지 선반입이 일어났고, L2에서는 13~24번 블록까지 선반입이 일어났으나, 캐쉬의 크기로 인해 실제 20~24번까지의 블록만 저장이 된다. 즉, 현재 시점에서 7~24 블록까지는 정확하지 못한 선반입 예측으로 인해 낭비되는 블록이라 볼 수 있다. 이러한 액세스 패턴이 이어진다면 하위레벨(L2) 캐쉬와 디스크에서는 엄청난 캐쉬 공간과 디스크 I/O의 낭비가 발생하게 된다.

그림 2의 예에서 알 수 있듯이, 다단계 저장장치의 선반입 기법은 기존 1단계 시스템에 비해 좀 더 정확한 예측에 기반하여 수행하여야 한다. 본 연구에서는 상위 시스템의 선반입 기법에 의존하지 않는, 넓은 용도로 사용할 수 있는 일반적인 하위 레벨 선반입 기법을 제안하였다. 제안한 기법의 성능을 평가하기 위하여, 정밀한 2단계 저장장치 시뮬레이터를 제작하여, 1단계 시스템에서로 다른 4가지의 선반입 정책을 적용하여 실험하였다. 2가지의 널리 쓰이는 트레이스를 이용하여 실험한 결과, 본 연구에서 제안한 선반입 기법을 이용할 경우, 상위 시스템의 선반입 정책이 무엇이든 상관없이 좋은 성능을 나타냄을 보였다. 하위 레벨 시스템에 기존에 연구된 선반입 시스템을 사용하였을 때 보다, 본 연구에서 제안된 일반적인 선반입 시스템을 적용할 경우, 응답시간을 측정하였을 때 최대 35%, 평균 16.56%의 성능 향상을 확인할 수 있다.

본 논문은 다음과 같은 구성으로 이루어진다. 이어지는 장에서는 기존의 선반입 시스템에 대해 간략히 알아보고, 3장에서는 PMS 알고리즘의 상세한 내용을 살펴본다. 4장에서는 시뮬레이터에 대한 소개와 실험 결과에 대한 분석이 이어지고, 마지막으로 결론과 향후 연구에 대해 다루게 된다.

2. 관련 연구

한 시스템 내에서의 선반입 기법은 많은 연구가 진행되어 왔다. 그러나 현재까지 다단계의 저장장치를 고려한 다단계 선반입 정책은 특별한 연구 결과가 없다.

한 시스템 내에서의 선반입 기법에 대한 많은 연구는 "언제", "무엇을" 선반입 할 것인가에 초점을 맞추어 왔다. 대부분의 선반입 연구는 순차적인 선반입을 기반으로 선반입 시기와 양을 효과적으로 결정하는데 주력하여 왔다. 일어난 요청에 연속적으로 이어지는 블록을 순차적으로 선반입을 하게 되면, 매우 작은 비용으로 디스크 접근을 할 수 있기 때문에 선반입 기법의 대부분은 순차적 접근을 하고, 본 연구에서 제안한 PMS 기법도 순차적 선반입 기반으로 효율적인 선반입 알고리즘을 개발하였다. 기존의 1레벨 시스템을 위한 선반입 연구는 다양하게 이루어져 왔으며, 본 연구에서는 그 중 가장 널리 쓰이는 선반입 정책인 AMP, SARC, 리눅스 선반입 정책, Read Ahead(RA) 정책을 사용하여 실험하고, PMS의 성능을 평가하였다.

최근 제안된 AMP[4] 선반입 정책은 "언제", "무엇을" 선반입 할지 결정하기 위하여, 선반입 등급(prefetch degree)와 요청 거리(trigger distance)라는 두 가지 값을 히스토리에 기반하여 동적으로 변화시킨다. 선반입 등급은, 실제 요청에 이어 순차적으로 얼마나 많은 블록을 선반입 할지를 결정하고, 요청 거리는 언제 선반입을 수행 할지를 결정하는 값이다. [4]의 연구에서 제안한 기법은 여러 개의 어플리케이션이 서로 다른 요청을 동시에 보낼 때 효과적으로 선반입을 수행하기 위하여 선반입으로 인해 캐쉬에 저장된 블록에 대해서 선반입 등급과 요청 거리를 각각 기록하여, 효과적인 선반입을 수행한다.

SARC 정책[5]은 IBM에서 제안된 정책으로 널리 쓰이는 저장 장치인 IBM DS6000/8000에서 실제 효과적으로 사용되는 선반입 기법이다. SARC는 단순한 선반입이 아닌, 선반입 기법과 버퍼 캐쉬 교체 정책을 합한 통합적인 정책을 제안 하였다. 순차적인 요청과 무작위적인 요청을 따로 처리하기 위해 SEQ와 RANDOM이라는 LRU기반의 2개의 큐를 이용하여 선반입과 교체정책을 수행한다.

널리 쓰이는 운영체제인 리눅스에서는 Read-ahead group과 Read-ahead window라는 정보를 각각의 파일별로 관리하여 선반입의 정도를 결정하는 정책을 사용한다. Read-ahead group은 현재 요청으로 일어난 모든 선반입 블록을 저장하고, Read-ahead window는 현재 파일에서 과거에 일어났던 선반입 블록까지 저장하고 있다. 만일 Read-ahead group에 해당하는 블록에 대한 요청이 다시 일어나게 되면, 과거의 선반입이 성공적이었다고 판단하고, 앞으로는 더 많은(기존 선반입 블록 수 * 2) 선반입을 수행하게 된다. 단, 최신 버전인 리눅스 커널 버전 2.6에서는 최대 32블록까지의 선반입을 수행한다.

[6]의 연구에서는 매우 간단한 선반입 기법을 제안했는데, 그것은 모든 요청에 대해 단 한 개의 추가 블록을 선반입 하는 방법이었다. 이후 이 기법을 확장하여, 모든 요청에 대해 N개의 블록을 선반입 하는 방법이 제안되었다. 본 연구에서는 이를 Read Ahead(RA)라고 칭하고, 항상 4개의 선반입을 수행하는 기법을 성능 평가의 일환으로 사용하였다.

3. PMS 알고리즘

본 연구에서 제안하는 다단계를 고려한 선반입 기법

은 상위레벨 시스템의 선반입 정책, 혹은 캐쉬 교체 정책이 무엇이던 전체 시스템의 성능을 향상 시킬 수 있는 독립적인 기법이다. 서론에서 설명하였듯이, 다단계 저장장치에서 각 단계 시스템이 독립적인 선반입 정책을 사용한다면, 상위레벨에서의 낭비가 작더라도 하위레벨 시스템에선 상위 시스템의 선반입이 포함된 요청을 처리하며 추가적인 선반입을 수행하기 때문에 전체 시스템에서는 캐쉬 공간과 디스크 I/O 측면에서 큰 낭비를 초래할 수 있다. 반대로, 상위 시스템에서 앞으로 일어날 요청보다 너무 작은 선반입을 수행한다면, 하위레벨에서도 역시 충분하지 않은 선반입을 수행할 것이기 때문에, 전체 시스템의 측면에서 보면 선반입의 효과를 충분히 누리지 못한다. 그래서 본 연구에서 제안하는 PMS(Prefetch strategy for Multi-level Storage system)는 상위레벨의 선반입이 포함된 요청을 분석하여 이제까지의 히스토리를 바탕으로 최대한 적절한 시기에 적절한 양을 선반입 하는 기법이다. 즉, PMS 기법은 하위 레벨의 시스템에 적용되는, 상위레벨의 선반입에 독립적으로 항상 좋은 성능을 이끌어 낼 수 있는 선반입 기법이다. 이를 위해 PMS 기법에서는 각각의 요청에 대해 다음의 두 가지 동작 중 하나, 혹은 두 개의 동작을 취해 선반입을 행함으로써, 상위 레벨의 선반입 정책이 적절하지 못한 수행을 하더라도 하위레벨에서는 이를 최대한 효율적으로 수행한다.

- Bypass : 상위 레벨에서 온 요청 중 적중되지 않은 블록의 일부는 캐쉬에 저장하지 않고 바로 하위레벨, 혹은 디스크로 요청한다. 상위레벨의 캐쉬가 부족하다면, Bypass를 최소화 하여 향후 미스되는 블록을 빠르게 제공할 수 있다.
- Readmore : 요청 블록에 연속되는 선반입을 요청한다. 얼마나 많은 블록을 선반입 할지 결정은 이제까지의 요청에 대한 히스토리를 기반으로 결정한다.

PMS 기법에서는 어떤 블록을 Bypass하고, 어떤 블록을 Readmore할지를 결정하기 위해 Bypass큐와 Readmore 큐를 사용한다. 두 개의 큐에는 실제 데이타가 아닌, 블록 번호만이 저장되기 때문에 메모리 사용량은 극히 적다. 또한 두 개의 큐는 각각 전체 캐쉬 크기의 1/10에 해당하는 블록번호 만을 저장하며 이는 LRU 정책에 의해 교체된다. 다양한 워크로드를 이용한 PMS 실험 결과 큐의 크기를 전체의 10% 이상으로 설정을 하여도 큰 이득이 없음을 알 수 있었기 때문에 메모리 사용량의 최소화를 위하여 본 실험에서는 Bypass와 Readmore 큐 모두 전체 캐쉬 크기의 10%로 설정 하였다. Bypass 큐에는 이전까지 bypass되었던 블록 번호가 저장되어 이후 상위레벨에서 요청된 블록이 bypass 큐에 있는 것이 확인된다면, 이는 상위 레벨 캐쉬의 크기가 부족하여 이전 선반입 된 블록이 사용되기 전에 교체되었다고 판단하여 앞으로의 bypass 블록 개수를 줄인다. Readmore 큐에는 readmore가 일어난 블록 이후의 일정 개수 블록 번호를 저장하여 이후 요청이 readmore 큐에 있는 블록 중 하나라면 readmore가 효과적으로 수행되고 있다는 증거이므로 readmore를 적극적으로 수행한다.

그림 3은 PMS 알고리즘의 동작을 설명하기 위한 간단한 예이다. 상위 레벨(L1) 시스템에서 1~5의 블록에 대한 요청이 들어왔을 때, PMS는 이제까지의 히스토리에 기반한 분석을 하여 bypass와 readmore 블록의 개수를 정한다. 위의 예에서는 1~3번 블록까지는 Bypass를 하고, 6~8블록은 선반입을 위한 Readmore를 수행하게 된다. 이어지는 9~11블록은 실제로 선반입이 수행되지 않고, Readmore 큐에 블록번호만 저장되어 향후 PMS의 동작에 도움이 되는 정보로 기록한다. PMS의 상세한 알고리즘은 그림 4의 의사코드를 통해 설명하였다.

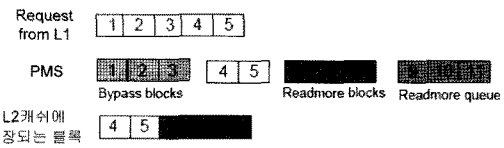


그림 3 PMS 알고리즘의 동작 예

PMS 정책에서는 상위 레벨의 선반입 정책이 너무 많은 선반입을 요청하여 하위 레벨에서 캐쉬 공간과 디스크 I/O의 낭비가 일어난다고 판단하면, Bypass 블록 수를 늘여서 낭비를 최소로 줄이고, 반대로 상위의 선반입 정책이 너무 작은 블록만을 선반입 한다고 판단되면 Readmore 블록을 늘여서 좀 더 적극적인 선반입을 수행한다.

4. 실험 환경 및 결과

4.1 시뮬레이터 개요

본 연구에서 제안한 PMS정책의 성능 평가를 위해 본 연구진은 트레이스를 입력으로 받아 2단계 저장 장치를 정밀하게 흉내 낸 시뮬레이터를 제작 하였다. 선반입 기법의 효과를 측정할 때 버퍼 캐쉬의 적중률(Hit ratio)만을 측정해서는 디스크의 혼잡도가 미치는 영향을 측정할 수가 없다. 선반입의 효과를 정확히 측정하기 위해서는 적중률뿐 아니라 각 요청을 실제로 처리하는데 걸리는 시간인 응답시간에 대한 측정이 이루어져야 한다. 응답시간을 측정하기 위해 본 연구에서는 네트워크 비용과 디스크 처리시간을 계산하였다. 상위레벨과 하위레벨 시스템이 네트워크로 연결되어 있기 때문에,

Algorithm 1: $PMS\_Process\_Req(req_u = [start_u, end_u])$

```
req_size = end_u − start_u + 1;
avg_req_size = average request size so far.
/* If req_size is larger than two times of avg_req_size, exclude it from calculation of
avg_req_size. */
rm_size = MAX(req_size, avg_req_size);
PMS_Set_Param(req_u);

start_pfc = start_u + bypass_length;
end_pfc = end_u + readmore_length;
process request [start_u, start_pfc − 1] directly;
forward request [start_pfc, end_pfc] to native L2 processing;

/*Insert new items into queues*/
if bypass_queue or readmore_queue is full then
    evict oldest items until required space is available;
insert [start_u, start_pfc − 1] into bypass_queue;
end_rm = end_pfc + rm_size;
insert [end_pfc, end_rm] into readmore_queue;
```

Algorithm 2: $PMS\_Set\_Param(req_u = [start_u, end_u])$

```
hit_cache = hit_bypass = hit_readmore = false;

/* Check against aggressive L1/L2 prefetching */
if ((req_size > avg_req_size) and (L2 cache is full)) then
    readmore_length = 0;
endif
if ([end_u, end_u + req_size] ∈ cache) then
    bypass_length = req_size;
    readmore_length = 0;
    return;
endif

/* Check hit status of L2 cache and PMS queues */
for start_u ≤ x ≤ end_u do
    if x ∈ cache then hit_cache = true;
    if x ∈ bypass_queue then hit_bypass = true;
    if x ∈ readmore_queue then hit_readmore = true;
endfor

/* Adjust PMS parameters */
if !hit_bypass then bypass_length + +;
if !hit_cache then
    if hit_bypass then bypass_length − −;
    if hit_readmore then
        readmore_length = rm_size;
    else readmore_length = 0;
endif
```

그림 4 PMS 알고리즘

네트워크에서 걸리는 시간을 수식적으로 계산하기 위하여 [7]에서 제안된 수식을 사용하였다. 이는 네트워크 전송이 일어날 때마다 발생하는 상수 비용(A)와 전송되는 데이타의 양에 따라 달라지는 변수 비용(B)를 이용하여 <A + B*데이타 길이>로 계산하였다. A와 B를 위해 1GBps 네트워크 카드를 장착한 LAN환경에서의 TCP/IP 패킷 전송 실험을 수행하였으며, 실험을 통해 발견한 값인 6ms와 0.03ms/block를 A와 B로 각각 설정하였다. 또한 디스크 드라이브를 흉내내기 위해 [8]에서 개발하고, 기존의 많은 연구에서 사용되었던 Disksim 2.0을 시뮬레이터에 부착하여 정밀하게 디스크 비용을 측정하였다. 본 연구에서는 Disksim 에서 제공하는 SCSI 인터페이스의 Seagate Cheetah 9LP 하드 디스크 드라이브 시뮬레이터를 실험에 사용하였다.

4.2 실험 트레이스

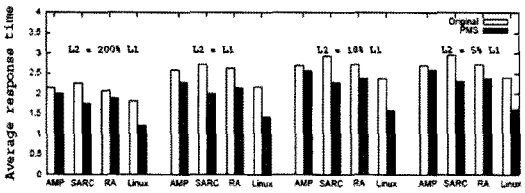
본 실험에서는 대형 저장장치 시스템에 널리 사용되는 벤치 마크 프로그램에서 수집한 두 가지의 현실적인 트레이스를 사용하였다. 첫째는 Storage Performance Council(SPC) 기관에서 수집하고, 여러 기존 여러 연구[9,10]에서 사용되었던 Web search 트레이스이다. Web search 트레이스는 실제 인터넷 서치 엔진에서 수집한 트레이스로서 74%의 요청이 무작위 요청이고, 소수만이 순차적인 요청으로 이루어진 트레이스이다. Web search의 풋프린트(foot print)는 총 8,392MB이다.

두 번째는 퍼듀 대학에서 2005년 수집된 트레이스로 cscope, gcc와 viewperf의 세 개의 프로그램을 동시에 수행하며 일어나는 모든 파일 I/O를 수집한 것이다[11]. 여기에는 총 12,514개의 파일에 대한 접근이 담겨있으며 총 풋프린트(foot print)는 792MB이다. 본 연구에서는 상위 레벨 시스템(L1)의 캐쉬 크기를 트레이스 풋프린트의 1%로 고정하고, 하위 레벨 시스템(L2)의 캐쉬 크기를 4가지 서로 다른 크기로 설정하여 실험 하였다. 단, 디스크를 흉내내기 위해 사용한 Disksim 2.0에서 지원하는 가장 큰 디스크의 크기가 9.1GB(Seagate Cheetah 9LP)에 불과 하기 때문에 Web search 트레이스는 처음 9.1GB에 해당하는 트레이스만을 실험에 사용하였다.

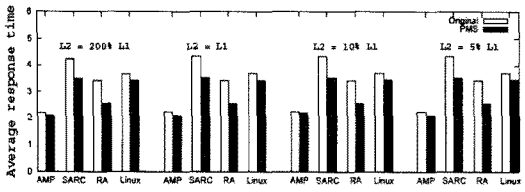
4.3 실험 결과

본 연구에서는 PMS의 성능 향상을 확인하기 위하여 기존에 널리 쓰이는 4가지의 선반입 기법을 구현하였다. 상위 레벨(L1)에는 AMP, SARC, Read Ahead, Linux 선반입의 4개의 기존 선반입 기법을 사용하고, 하위 레벨(L2)에는 PMS기법을 사용하였다. PMS의 성능을 비교하기 위한 비교 대상으로는 상위 레벨과 같은 선반입 기법을 하위 레벨 시스템에 적용하여 실험을 통해 성능을 측정하였다. 또한, 하위 레벨 시스템은 일반적으로 여러 개의 상위 레벨 시스템의 요청을 동시에 받을 수 있으므로 상황에 따라 가용한 캐쉬의 크기가 달라진다. 그래서 본 실험에서도 하위 레벨 시스템의 캐쉬 크기를 서로 다른 4개로 구분하여 실험하였다. 앞서 설명 하였듯이 상위 레벨 시스템(L1)의 캐쉬 크기는 트레이스 풋프린트(foot print)의 1%로 고정하였고, 하위 레벨 시스템(L2)의 캐쉬 크기는 상위 시스템의 200%, 100%, 10%, 5%의 4가지로 구성하여 각각 실험을 하였다. 일반적으로 하위 레벨 시스템의 메모리 크기는 상위 시스템보다 큰 것이 일반적이므로 하위 레벨의 캐쉬 크기는 상위 시스템의 200%인 경우도 실험 군에 포함하였다.

그림 5에서는 하위 레벨 시스템에 L1과 동일한 선반입 기법을 사용하는 기존 선반입 기법(original)과 PMS를 적용한 시스템의 평균 응답시간(Average response time)을 측정하여 그래프로 보였다. 그림 5의 (a)는 Web search 트레이스를 사용하여 4개의 서로 다른 선반입 기법과 4개의 L2 캐쉬 크기에 따른 요청별 평균 응답시간을 나타내었고, (b)의 그래프는 Multi 트레이스

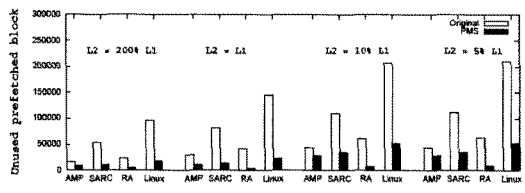


(a) Web search 트레이스의 평균 응답시간(ms)

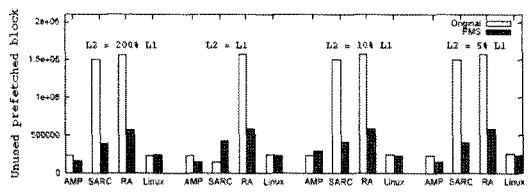


(b) Multi 트레이스의 평균 응답시간(ms)

그림 5 기존 선반입 기법과 PMS의 성능 비교



(a) Web search 트레이스의 낭비된 선반입 블록



(b) Multi 트레이스의 낭비된 선반입 블록

그림 6 기존 선반입 기법과 PMS의 낭비된 블록 비교

의 경우를 나타내었다. 그래프에서 확인할 수 있듯이, PMS 선반입 기법을 하위 레벨 시스템에 적용할 경우 캐쉬의 크기나 상위 레벨 선반입 기법에 무관하게 모든 경우에 평균 응답시간을 줄일 수 있음을 알 수 있다.

PMS 선반입 기법이 기존의 선반입 정책에 비해 가장 큰 성능 향상을 보인 경우는 Web search 트레이스에서 L2 캐쉬의 크기가 L1 캐쉬의 크기와 같은 상황이고, L1 시스템이 Linux 선반입 기법을 사용했을 경우, 최대 35%의 성능 차이가 나타남을 알 수 있다. 전체 32종류의 실험 모두에서 PMS는 기존 정책에 비해 향상된 성능을 보였고, 향상폭은 평균 16.56%로 나타났다.

그림 6에서는 PMS와 기존 선반입 기법의 낭비된 선반입 블록의 개수를 비교하였다.

그래프에서 쉽게 확인할 수 있듯이, PMS 기법의 경우 거의 대부분의 실험에서 낭비되는 선반입 블록의 개수를 크게 줄일 수 있음을 알 수 있다. PMS 기법은 현재까지의 히스토리를 기반으로 Bypass 블록과 Read-more 블록을 동적으로 결정하기 때문에, 상위 블록에서 너무 많거나 너무 적은 선반입을 수행했을 경우에도, 하위 레벨의 선반입 정도를 적절하게 결정 할 수 있다. 그렇기 때문에, 대부분의 경우 기존의 선반입 기법을 하위 레벨에 적용한 경우에 비해 사용되지 않고 낭비되는 선반입을 줄일 수 있다.

지면의 부족으로 인해 하위레벨 캐쉬의 적중률과 디스크에서 실제 처리한 양을 나타내는 상세한 자료를 본지에 소개할 수는 없지만, 적중률과 디스크 요청량에서도 PMS는 기존 선반입 정책에 비해 대체로 우월한 성능을 보였다.

4. 결론 및 향후 연구 과제

본 연구에서는 다단계로 구성된 저장 장치 시스템에서 하위 단계에 적용하여 효과적으로 선반입을 수행할 수 있는 PMS(Prefetching Strategy for Multi-level Storage system) 기법을 제안하였다. 이는 상위 시스템의 선반입 정책이 부정확하여 너무 많거나 너무 적은 선반입을 수행하였을 때, 하위 레벨 시스템에서 선반입의 부작용을 최소화하고, 전체 시스템의 성능을 최대화하기 위하여, 히스토리를 기반으로 선반입을 결정하는 정책이다. 2단계 저장장치 시스템을 흉내낸 시뮬레이터와 대형 서버 시스템에서 널리 쓰이는 2가지 트레이스를 이용하여 성능 측정을 한 결과, PMS 선반입 기법은 기존 선반입 정책을 하위 레벨에 그대로 사용한 시스템에 비해 모든 경우에서 큰 성능 향상을 나타내었다. 요청별 평균 응답시간은 물론, 사용되지 않고 낭비된 선반입 블록의 개수에서도 거의 대부분 좋은 성능을 나타내었다.

향후 본 연구팀은, 더욱 향상된 선반입 기법을 위해 몇 가지 향후 연구 과제에 대한 계획을 세웠다. 첫째, 현재의 실험 환경은 2단계 저장 장치만을 사용하였기 때문에, 좀 더 많은 단계로 시스템이 구성되었을 경우에 좀 더 효율적인 선반입 정책에 대한 연구를 할 예정이다. 두 번째로, 본 연구에서는 2가지의 트레이스만을 이용하여 실험을 하였는데, 좀 더 다양하고 커다란 트레이스를 이용하여 선반입 정책의 성능을 좀 더 향상시키기 위한 연구를 진행할 예정이다. 마지막으로, 본 연구에서 제안된 PMS 선반입 정책을 실제 다단계 시스템의 하위 레벨에 직접 적용하여 실제 환경에서는 얼마나 큰 성능 향상을 보이는지 실제로 확인할 계획을 가지고 있다.

참 고 문 헌

[1] Zhifeng Chen, Yuanyuan Zhou, and Kai Li. Eviction-based cache placement for storage caches. In *Proceedings of the 2003 USENIX Annual Technical Conference*, 2003.

[2] Song Jiang and Xiaodong Zhang. ULC: a file block placement and replacement protocol to effectively exploit hierarchical locality in multi-level buffer caches. In *Proceedings of the 24th International Conference on Distributed Computing Systems(ICDCS)*, 2004.

[3] Theodore M. Wong and John Wilkes. My cache or yours? making storage more exclusive. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, 2002.

[4] B. Gill and L. Bathen. AMP: Adaptive multi-stream prefetching in a shared cache. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies(FAST)*, 2007.

[5] B. Gill and D. Modha. SARC: Sequential prefetching in adaptive replacement cache. In *Proceedings of the 2005 USENIX Annual Technical Conference*, pp. 293-308, 2005.

[6] A. Smith. Cache memories. In *ACM Computing Surveys(CSUR)*, Vol.14, pp. 473-530. ACM Press, 1982.

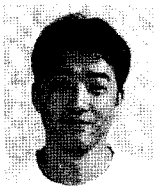
[7] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. Logp: towards a realistic model of parallel computation. In *Proceedings of the fourth ACM-SIGPLAN symposium on Principles and practice of parallel programming(PPOPP)*, 1993.

[8] G. Ganger, B. Worthington, and Y. Patt. The disksim simulation environment version 2.0, Dec. 1999.

[9] Zhenmin Li, Zhifeng Chen, Sudarshan M. Srinivasan, and Yuanyuan Zhou. C-Miner: Mining block correlations in storage systems. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies(FAST)*, 2004.

[10] Qingbo Zhu, Zhifeng Chen, Lin Tan, Yuanyuan Zhou, Kimberly Keeton, and John Wilkes. Hibernator: helping disk arrays sleep through the winter. In *Proceedings of the twentieth ACM symposium on Operating systems principles (SOSP)*, 2005.

[11] Ali R. Butt, Chris Gniady, and Y. Charlie Hu. The performance impact of kernel prefetching on buffer cache replacement algorithms. In *Proceedings of the International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS)*, 2005.



이 규 형

2003년 홍익대학교 정보컴퓨터공학부(학사). 2008년 홍익대학교 컴퓨터공학과(석사). 2008년~현재 퍼듀대학교 컴퓨터공학과 박사과정. 관심분야는 운영체제, 차세대저장장치, 페러럴시스템

이 효 정

2001년 홍익대학교 정보컴퓨터공학부(학사). 2000년~2005년 ㈜이칼로스 연구원 2005년~2006년 ㈜웨폴리닷넷 연구원 2007년~현재 홍익대학교 컴퓨터공학과 석사과정. 관심분야는 운영체제, 내장형 시스템, 차세대저장장치

노 삼 혁

1986년 서울대학교 컴퓨터공학과 학사 1993년 메릴랜드대학교 컴퓨터과학과 박사. 1993년~1994년 조지워싱턴대학교 객원 조교수. 1994년~현재 홍익대학교 정보컴퓨터공학부 교수. 관심분야는 운영체제, 플래시메모리소프트웨어, 내장형시스템, 차세대저장장치