최근성과 참조 횟수에 기반한 페이지 교체 기법

이 승 훈[†] · 이 종 우[††] · 조 성 제[†††]

요 약

운영체제의 가상 메모리 시스템에 적용할 페이지 교체 정책은 요구 페이징 시스템의 성능에 큰 영향을 미친다. 대표적인 메모리 페이지 교체 정책으로는 LRU와 LFU가 있다. LRU 정책은 많은 경우에 좋은 성능을 보이며 시스템 부하 변화에 잘 적응하지만, 자주 참조되는 페이지와 가끔 참조되는 페이지를 구별하지 못한다. LFU 정책은 참조 횟수가 가장 작은 페이지를 교체하는 기법으로, 과거의 모든 참조를 반영하지만 이전에 참조된 페이지와 최근에 참조된 페이지를 식별하지 못한다. 따라서 LFU는 변화하는 작업 부하에 잘 적응하지 못한다. 본 논문에서는 먼저 8개의 응용에 대해 메모리 참조 패턴을 분석하여 보았다. 그 참조 패턴을 보면 어떤 경우에는 최근에 참조된 페이지가 계속 참조되며, 또 다른 경우에는 자주 참조되는 페이지가 계속 참조되는 경향이 있다. 즉, 응용에 의해 참조되는 메모리 페이지는 최근성과 참조 횟수 모두에 의해 가치가 결정되며, LRU나 LFU 정책 한 가지만으로는 페이지 교체 정책을 최적화하기 어렵다. 따라서 본 논문에서는 LRU 기법과 LFU 기법을 결합한 새로운 교체 기법을 제안한다. 제안한 기법에서는 페이지 리스트를 LRU 리스트와 LFU 리스트를 나누어 관리하는데, 이 두 리스트에서는 각각 최근성과 참조 횟수를 기반으로 페이지 리스트 순서가 유지된다. 과거에 자주 참조되었던 페이지가 LRU 정책에 의해 교체되어 빠져나가는 경우를 LFU 정책 병행 사용을 통해 줄임으로써, 최근성 가치에 의해 참조 횟수 가치가 훼손되는 경우를 줄인다. 트레이스-기반 시뮬레이션 결과, 제안 기법이 이전에 알려진 페이지 교체 기법보다 좋은 성능을 보일 때가 있음을 확인하였는데, 특히, 과거에 자주 참조했던 페이지를 일정 시간 경과한 후에 다시 참조하는 패턴을 보이는 응용들에서 제안 기법이 기존의 기법들보다 우수하다는 것을 알 수 있었다.

A Page Replacement Scheme Based on Recency and Frequency

Seung-Hoon Lee[†] · Jongwoo Lee[††] · Seongje Cho[†††]

ABSTRACT

In the virtual memory system, page replacement policy exerts a great influence on the performance of demand paging. There are LRU(Least Recently Used) and LFU (Least Frequently Used) as the typical replacement policies. The LRU policy performs effectively in many cases and adapts well to the changing workloads compared to other policies. It however cannot distinguish well between frequently and infrequently referenced pages. The LFU policy requires that the page with the smallest reference count be replaced. Though it considers all the references in the past, it cannot discriminate between references that occurred far back in the past and the more recent ones. Thus, it cannot adapt well to the changing workload. In this paper, we first analyze memory reference patterns of eight applications. The patterns show that the recently referenced pages or the frequently referenced pages are accessed continuously as the case may be. So it is rather hard to optimize page replacement scheme by using just one of the LRU or LFU policy. This paper makes an attempt to combine the advantages of the two policies and proposes a new page replacement policy. In the proposed policy, paging list is divided into two lists (LRU and LFU lists). By keeping the two lists in recency and reference frequency order respectively, we try to restrain the highly referenced pages in the past from being replaced by the LRU policy. Results from trace-driven simulations show that there exists points on the spectrum at which the proposed policy performs better than the previously known policies for the workloads we considered. Especially, we can see that our policy outperforms the existing ones in such applications that have reference patterns of re-accessing the frequently referenced pages in the past after some time.

키워드 : 운영체제(Operating System), 가상메모리(Virtual Memory), 페이지교체(Page Replacement), 성능평가(Performance Evaluation)

1. 서 론

운영체제의 가상 메모리 관리자는 일반적으로 현재 사용중인 프로세스 부분들은 주기억장치 안에 적재시키고 사용중이지 않은 부분들은 디스크에 저장하고 있어서 물리적 메모리가 비효율적으로 사용되는 것을 막는다. 이러한 방법 중 하나인 요구 페이징 기법은 가상 메모리와 물리적 메모리를 페이지라는 작은 조각으로 나눈 후, 가상 페이지들이 접근되는 경우에만 주기억장치에 읽어들이는 기법이다.

요구 페이징 시스템에서 수행 중인 프로세스가 현재 주기억장치에 없는 페이지를 참조하면 페이지 폴트(page fault)가 발생한다. 그러면 운영체제는 자유 페이지 프레임(free page

※ 이 연구는 2000년도 단국대학교 대학연구비의 지원으로 연구되었음.
† 정 회 원 : 단국대학교 대학원 전산통계학과
†† 정 회 원 : 한림대학교 정보통신공학부 교수
††† 정 회 원 : 단국대학교 전산통계학과 교수
논문접수 : 2001년 6월 9일, 심사완료 : 2001년 10월 11일

frame)을 할당하여 폴트를 유발한 페이지를 반입하게 된다. 이때 자유 페이지 프레임이 없다면 페이지 교체가 발생하게 된다. 이처럼 페이지 폴트를 처리하기 위해 최악의 경우에는 디스크 접근이 두 번 필요하게 되며, 이는 시스템 성능을 저하시키는 주요 요인이다. 따라서 페이지 폴트 수를 줄이기 위해 효율적인 페이지 교체 기법이 필요하며, 페이지 교체 기법은 다시 참조될 가능성이 큰 페이지들을 주기억장치에 계속 유지시켜 페이지 적중율을 높이고자 한다. 페이지 교체 기법으로 많이 사용되는 것이 LRU와 LFU 기법이다.

LFU 기법은 각 페이지들이 얼마나 많이 참조되어 졌는지에 착안한다. 페이지 교체가 필요할 경우 이 기법은 참조 횟수가 가장 작은 페이지를 교체한다. 이 기법은 참조패턴이 변하지 않는 안정적인 작업부하(workload)에서는 좋은 성능을 보이는 반면 작업부하가 변하는 경우 이에 적응하지 못하여 성능이 떨어진다. LRU 기법은 가장 오랫동안 참조되지 않은 페이지를 교체한다. 순수 LRU 기법을 구현하기 위해서 각 페이지들이 참조될 때마다 그때의 시간을 페이지 테이블에 기억시켜야 한다. 이 작업은 막대한 오버헤드를 초래하게 되며 따라서 거의 사용되지 않는다. 대신 더 작은 오버헤드를 가지는, LRU에 근사하는 기법이 사용된다. 그리고 LRU 기법은 마지막 참조의 최근성만을 기준으로 페이지를 교체하기 때문에 페이지 참조 횟수를 반영하지 못한다.

본 논문에서는 먼저 8개의 응용에 대해 메모리 참조 패턴을 분석해 보았다. 일부 응용을 제외한 대부분의 응용에서 최근에 참조된 페이지가 계속 참조되는 경향이 있으며, 또한 자주 참조된 페이지도 계속 참조되는 경향이 있다. 이는 순수한 LRU나 LFU 페이지 교체 기법이 최적의 페이지 교체 기법이 아니라는 것을 의미한다. 본 논문에서는 페이지의 최근성과 참조 횟수를 동시에 고려할 수 있도록 페이징 리스트를 LRU 리스트와 LFU 리스트로 분할하여 관리하는 새로운 페이지 교체 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 참조의 지역성과 교체 기법들에 대해 설명하고, 3장에서는 실험에 사용된 8가지 응용 트레이스들의 페이지 참조 패턴에 대해 설명하고, 4장에서는 제안 기법에 대해 설명하고 있으며, 5장에서는 8가지 응용 트레이스들에 대한 다양한 실험 결과들을 통해 제안 기법의 성능을 설명한다. 마지막으로 6장에서 결론을 내린다.

2. 관련 연구

운영체제의 기억장치 관리에서 많이 나타나는 특성인 지역성(locality)은 시간과 공간적인 지역성으로 나뉜다. 이것은 이론적인 특성이 아니라 관측된 실험적인 특성이라고 할 수 있다. 예를 들어 페이징 시스템에서 프로세스들은 각각 그들의 페이지들 중 소수의 몇몇 페이지들을 특히 많이 사용하는 경향이 있으며 이 몇몇 페이지들은 그 프로세스의 가상 주소공간 내에서 서로 인접해 있는 경향이 많다. 이것은 각 프로세스가 어떤 주어진 시간동안 그의 전 페이지들 중 특별한 몇 개의 페이지 내에서 실행되는 경향이 있음을 의미한다. 시간 지역성은 최초로 참조된 기억장소가 가까운 미래에도 계속 참조될 가능성이 높음을 의미한다. 예를들면, 순환(looping), 부 프로그램(subroutine), 스택에 사용되는 변수들의 경우이다. 공간 지역성은 일단 하나의 기억장소가 참조되면 그 근처의 기억장소가 계속 참조되는 경향이 있음을 의미한다. 예를 들면, 배열순례(array traversal), 순차적 코드의 실행의 경우이다. 지역성의 가장 중요한 결과로는 각 프로그램이 많이 참조하는 페이지들이 주 메모리에 있는 한 그 프로그램은 효율적으로 실행될 수 있다는 것이다.

지역성을 반영하는 페이지 교체 기법 외에, 페이지 참조 패턴에 따라 최적의 페이지 교체 기법을 적용하고자 하는 연구들이 수행되어 왔다. SEQ 기법[6]은 정상적으로는 LRU 기법을 사용하다가 순차적인 참조에서 페이지 폴트가 오랫동안 나타나면 그것을 감지하고, 그때 MRU 기법을 사용한다. 순차적인 참조란 증가 혹은 감소하는 방향으로 인접한 가상 주소 페이지들의 참조가 일어나는 것을 말한다. 순차적인 참조에서 오랫동안 페이지 폴트를 보이는 응용들에 대해서는 LRU 기법보다 좋은 성능을 보여주고, 일반적인 응용들에 대해서는 LRU 기법과 비슷한 성능을 보인다.

EELRU 기법[10]은 SEQ 기법과 같이 정상적으로는 LRU 기법을 사용하다가 최근성 분포(recency distribution)를 기반으로 하여 주기억장치보다 큰 반복 패턴을 발견하고, LRU 리스트에서 페이지의 교체 위치를 변화시킴으로써 반복 참조의 적중률을 높였다. 이 기법의 구조는 매우 간단하다. 최근에 참조된 많은 페이지들이 교체되지 않았다면, LRU 기법을 수행한다. 반대의 경우에는 fallback 기법을 적용한다. fallback 기법은 가장 오랫동안 사용되지 않은 페이지를 교체하거나, 최근성 분포에서 e번째로 가장 최근에 사용된 페이지를 교체한다. e라는 인자는 미리 결정된 최근성에 관련된 위치이다.

LRFU 기법[1]은 참조 횟수와 참조 최근성 모두를 이용하는 파일 버퍼 캐시 블록 교체 기법이다. 이 기법에서는 블록의 재참조 가능성을 블록의 가치로 표현한다. 이 기법의 가장 기본적인 동작은 블록에 대한 참조가 발생하면 블록의 가치를 증가시키며, 시간이 지남에 따라 이러한 블록의 가치를 감소시키는 것이다. 하지만 [1]에서 대상으로 삼은 것은 메모리 참조가 아닌 파일 블록 참조이다. 따라서 본 논문에서 대상으로 하는 가상 메모리 시스템의 메모리 참조는 차이가 있다 하겠다.

3. 페이지 참조 패턴

3.1 트레이스의 구성

<표 1>은 리눅스 시스템 상에서 8개 응용들을 대상으로

VMTrace를 사용해 생성한 가상메모리 트레이스를 보이고 있다. VMTrace와 링크된 응용들이 실행되면서 페이지들을 참조할 때마다 VMTrace는 참조된 페이지의 번호를 출력한다. 트레이스들은 이러한 출력된 페이지 번호들로 이루어져 있다. 참고로 <표 1>의 트레이스는 텍사스 주립 대학(오스틴)에서 다운로드받은 것이다.

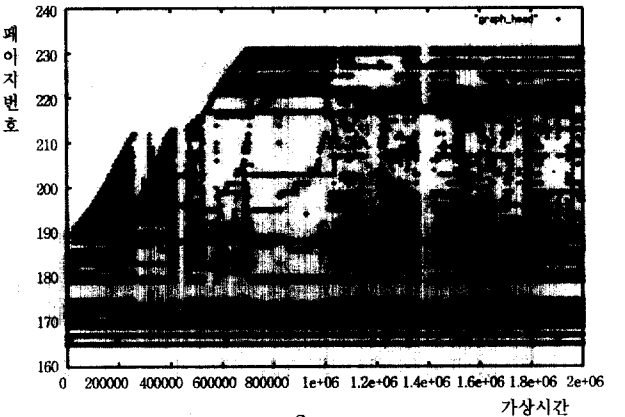
〈표 1〉 트레이스의 구성

트레이스로 사용된 응용	각 응용에 대한 설명	트레이스 크 기	Working Set 크기 (4KB 페이지 수)
p2c	Pascal->C transformer	30,722,431	132
lindsay	hypercube simulator	123,690,749	521
gs3.33	GhostScript, a software PostScript interpreter	134,371,942	558
grobner	grobner calculated Grobner basis functions	7,787,835	67
gcc-2.7.2	GNU C/C++ compiler	37,524,334	458
espresso	circuit simulator	326,938,361	77
gnuplot	the GNU plotting utility	68,458,509	7,718
rscheme	implementation of Scheme	151,606,733	2,039

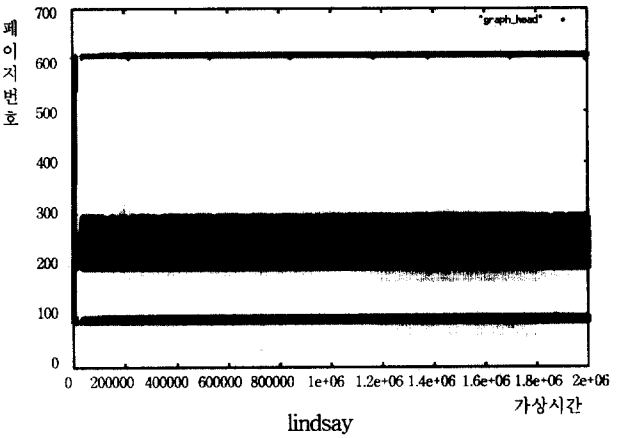
3.2 페이지의 참조패턴

각 응용의 메모리 참조 패턴을 분석하기 위해 본 논문에서는 우선 8개 응용 트레이스에 대해서 초기 200만 페이지 참조들의 패턴을 그래프로 표시해 보았다. (그림 1)은 8개의 응용 트레이스 각각에 대한 페이지 참조 패턴을 보이고 있다. 이 그래프에서 x축은 가상시간을 나타내는데, 최초로 페이지 참조가 일어났을 때를 가상시간 1로 보고 두 번째 페이지 참조가 일어났을 때를 가상시간 2라고 본다. y축은 해당되는 가상시간에 참조되는 페이지 번호를 나타낸다. 이 결과에 의하면 일부 응용(lindsay와 gnuplot)을 제외한 대부분의 응용들에서는 최근에 참조된 페이지가 강한 지역성을 보임과 동시에 자주 참조된 페이지가 계속 참조되는 패턴을 보인다는 것을 알 수 있다.

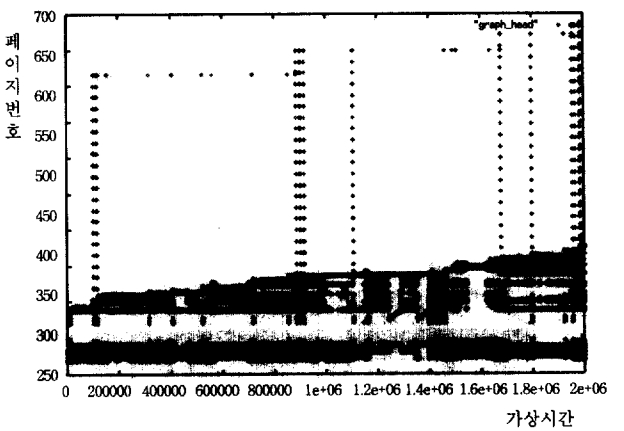
예를 들어, (그림 1)에서 p2c와 gs, 그리고 grobner는 최근에 참조된 페이지가 계속 참조되는 성향과 자주 참조된 페이지가 계속 참조되는 성향을 모두 갖고 있음을 알 수 있다. 또한, 그래프의 모양으로 보아 p2c 보다는 gs와 grobner가 이러한 성향을 좀 더 강하게 띄고 있음도 알 수 있다. 한편, espresso의 페이지 참조 패턴에서는 한 번 참조된 페이지가 계속 참조되는 부분과 페이지가 바뀌면서 참조되는 부분이 발견된다. gnuplot에서는 순차적인 참조와 지역적인 참조가 확연히 구별되고 있음을 알 수 있으며 전체 페이지 참조 스트림을 분석해보면 반복 참조 패턴도 나타난다. rscheme 응용의 페이지 참조 패턴은 최근에 참조된 페이지가 계속 참조되고 자주 참조된 페이지가 계속 참조되는 경향을 보여준다.



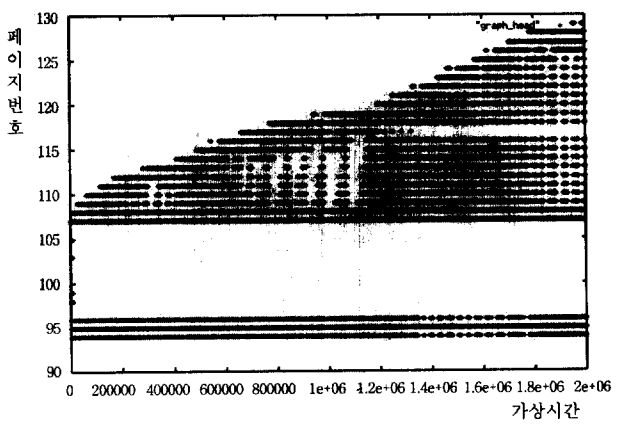
p2c



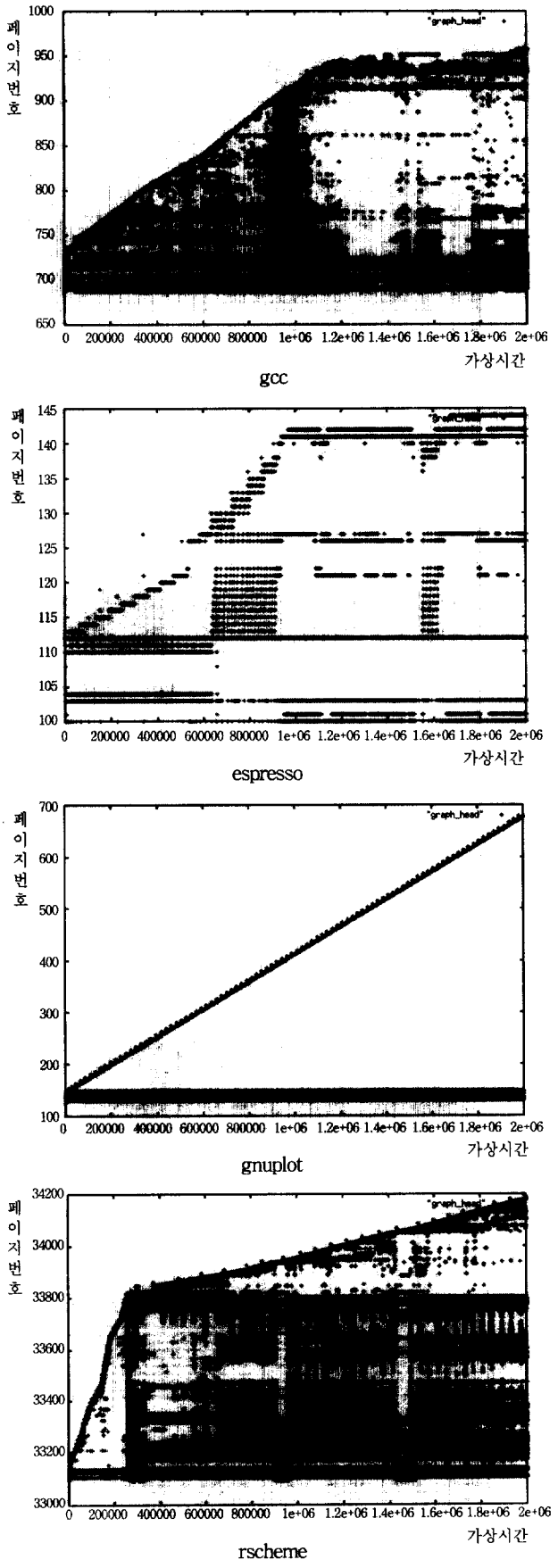
lindsay



gs



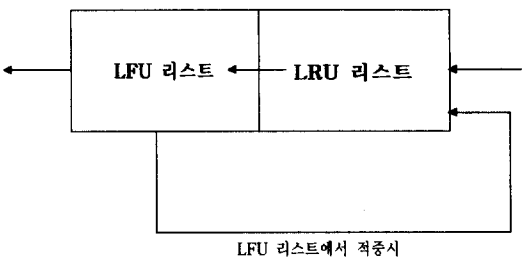
grobner



(그림 1) 8개 응용의 참조 패턴 추적 결과

4. 제안 기법

참조 패턴들을 분석한 결과 페이지의 최근성과 참조 수, 이 두 가지 정보가 동등하게 중요해져 두 정보 모두를 고려한 새로운 교체 기법이 요구된다. 따라서, 페이지 참조의 최근성과 참조 횟수를 모두 고려한 새로운 페이지 교체 기법을 제안한다. 제안 기법의 구조는 (그림 2)와 같다. 페이징 리스트를 두 부분, 즉 $\frac{m}{n}$은 LRU 리스트, $\frac{(n-m)}{n}$은 LFU 리스트로 나누어 관리한다. 이 때 n은 페이징 리스트를 구성하는 전체 페이지의 수를 나타내며 $m \leq n$이다. 페이지가 처음 참조되면 LRU 리스트에 삽입되는데 LRU 리스트가 가득 차게 되면 가장 오랫동안 사용되지 않은 페이지(LRU 리스트의 뒤에 있는 페이지)를 LFU 리스트로 보낸다. 참조된 페이지가 LFU 리스트에 있으면 그 페이지는 다시 LRU 리스트로 이동한다. 두 리스트가 가득 찼을 때 페이지 폴트가 발생하면 먼저 LFU 페이지를 교체하고, LRU 페이지를 LFU 리스트로 이동시킨 다음 새로 참조된 페이지를 LRU 리스트의 앞에 삽입한다. 해당 페이지가 참조될 때마다 그 페이지의 참조 횟수는 증가한다. 교체된 페이지가 재참조될 경우 다시 LRU 리스트의 앞에 삽입하게 되는데, 이때도 이전의 참조 횟수가 누적되어 유지된다.



(그림 2) 제안 기법의 구조

자유 리스트가 없을시, LRU 리스트에서 히트될 경우에 현재 참조된 페이지를 LRU 리스트의 가장 앞에 삽입하고, LFU 리스트에서 히트될 경우에 LRU 리스트에서 가장 오랫동안 사용되지 않은 페이지를 LFU 리스트로 보내고 현재 참조된 페이지를 LRU 리스트의 가장 앞에 삽입한다. 자유 리스트가 없을시, 두 리스트 모두에서 미스일 경우에 LFU 리스트에서 페이지 참조횟수가 가장 작은 페이지를 교체하고, LRU 리스트의 가장 오랫동안 사용되지 않은 페이지를 LFU 리스트로 보내고, 현재 참조된 페이지를 LRU 리스트의 가장 앞에 삽입한다. (그림 3)은 제안 기법의 알고리즘을 나타낸다. (그림 4)는 제안 기법 안에서 수행되는, LRU 리스트에 페이지를 삽입하는 알고리즘을 나타낸다.

```
while (ref_page에 대한 참조가 일어남) {
  if (현재 참조된 페이지가 swap 리스트에 있음) {
      /* 교체된 페이지가 재참조 */
         페이지 폴트 수 증가;
         ref_page에 대한 참조 횟수 증가;
         swap 리스트에서 참조 페이지 제거;
         (그림 4)의 알고리즘 수행
  }
     /* LRU, LFU 리스트 둘 중 하나에 해당 페이지가 있는지
     조사 */
  else if (LRU, LFU 두 리스트에 해당 페이지가 없음) {  /* 최
  초 참조 */
         페이지 폴트 수 증가;
         ref_page에 대한 참조 횟수 = 1;
         (그림 4)의 알고리즘 수행
  }
  /* LRU, LFU 리스트 둘 중 하나에 해당 페이지 있음 */
  else if (현재 참조된 페이지가 LRU 리스트에 있음) {
         ref_page에 대한 참조 횟수 증가;
         ref_page를 LRU 리스트의 제일 앞으로 이동;
  }
  else { /* LFU 리스트에 해당 페이지가 있음 */
         ref_page에 대한 참조 횟수 증가;
         ref_page를 LFU 리스트에서 제거;
         (그림 4)의 알고리즘 수행
  }
}
```

(그림 3) 제안 기법의 알고리즘

```
insert_page_to_lru_list(ref_page)
{ /* 입력 : 현재 참조된 페이지 (ref_page) */
 if (LRU 리스트가 가득 차 있음) {
   if (LFU 리스트가 가득 차 있음) {
    LFU 리스트에서 참조 횟수가 가장 적은 페이지 교체;
    LRU 리스트의 가장 뒤에 있는 페이지를 LFU 리스트로 이
    동;
    ref_page를 LRU 리스트 가장 앞에 삽입;
   }
   else { /* LFU 리스트가 가득 차 있지 않음 */
    LRU 리스트의 가장 뒤에 있는 페이지를 LFU 리스트로 이
    동;
    ref_page를 LRU 리스트 가장 앞에 삽입;
   }
 }
 else /* LRU 리스트가 가득 차 있지 않음 */
   ref_page를 LRU 리스트 가장 앞에 삽입;
}
```

(그림 4) LRU 리스트에 페이지를 삽입하는 알고리즘

5. 실험 및 평가

5.1 트레이스 데이터를 사용한 시뮬레이션 실험 결과

3장에서 설명했던 각 응용의 트레이스 데이터에 대해 LRU 기법과 제안 기법, FIFO 기법, LFU 기법, 그리고 MRU 기

〈표 2〉 트레이스-기반 시뮬레이션을 통한 제안 기법의 성능 평가 실험 결과
(각 항목은 페이지 폴트 수. 1위 항목을 음영 처리했음.)

p2c - 전체 참조 횟수 : 30,722,431

교체기법 \ 프레임 수	50	60	70	80	90	100	110	120	132
LRU									132
제안 기법(5/6)	59266	36509	25349	15714	9134	5054	2535	1037	132
FIFO	80127	50529	32767	20736	12393	6693	3200	1388	132
LFU	1269826	1005350	764193	529021	327973	173923	72393	17647	132
MRU	2272151	1715737	1298869	941618	643519	426120	266790	110220	132

lindsay - 전체 참조 횟수 : 123,690,749

교체기법 \ 프레임수	150	200	300	400	450	500	521
LRU	633	633	633	633	523	523	521
제안 기법(5/6)				632			521
FIFO	637	636	635	634	525	525	521
LFU	673	673	673				521
MRU	5080075	1131094	314834	314834	314834	314834	521

gs - 전체 참조 횟수 : 134,371,942

교체기법 \ 프레임 수	100	200	250	300	350	400	500	558
LRU					961		590	558
제안 기법(5/6)	16715	2214	1407	1145		714		558
FIFO	24460	3473	2535	1885	1328	1140	809	558
LFU	1596250	292493	103275	29052	6497	2379	962	558
MRU	6817717	4613750	3772559	2779578	2012401	1337349	259817	558

grobner - 전체 참조 횟수 : 7787835							
프레임 수 / 교체기법	35	40	45	50	55	60	67
LRU							67
제안 기법(5/6)	7756	2048	611	176	83	73	67
FIFO	5304	1678	492	197	114	107	67
LFU	1048039	684286	378161	218306	74285	1552	67
MRU	181261	138016	89961	57736	29201	7853	67

gcc - 전체 참조 횟수 : 37524334							
프레임 수 / 교체기법	100	200	250	300	350	400	458
LRU							458
제안 기법(5/6)	35133	7800	3169	1630	1031	678	458
FIFO	42858	8522	4562	2786	1677	1088	458
LFU	1302959	262874	86380	31439	17331	9494	458
MRU	5260870	3867846	3074508	2325055	1660239	791638	458

espresso - 전체 참조 횟수 : 326938361							
프레임 수 / 교체기법	20	30	40	50	60	70	77
LRU				177	93	88	77
제안 기법(5/6)	62857	4293	1095				77
FIFO	70773	7026	1534	379	123	119	77
LFU	17074312	4196391	766689	113834	10798	466	77
MRU	5396835	4445873	4028025	3289320	1648156	693707	77

gnuplot -전체 참조 횟수 : 68458509							
프레임 수 / 교체기법	5000	5500	6000	6500	7000	7500	7718
LRU	23139	23139	23139	23139	23139	23139	7718
제안 기법(5/6)			21167	19443	18992		7718
FIFO	23162	23161	23159	23159	23159	23159	7718
LFU	23254	21882				13500	7718
MRU	7938824	6478513	4997795	3563170	2100422	637908	7718

rscheme -전체 참조 횟수 : 151606733							
프레임 수 / 교체기법	1000	1200	1400	1600	1800	2000	2039
LRU			10945	7442	4460	2250	2039
제안 기법(5/6)	21557	15161					2039
FIFO	31204	22141	14662	9641	5288	2273	2039
LFU	627931	376954	221205	94972	23830	2422	2039
MRU	59182285	46035548	33227219	20879963	8293282	637487	2039

법을 각각 적용해보았을 경우 발생하는 페이지 폴트의 수를 측정하였다. 제안 기법에서는 전체 페이지 리스트의 5/6를 LRU 리스트로, 1/6을 LFU 리스트로 관리하도록 하였다. <표 2>는 실험결과를 보이고 있다.

<표 2>에서 전반적으로 볼 때 본 논문에서 제안한 기법과 LRU, 또는 제안 기법과 LFU가 경쟁하고 있다는 사실을 알 수 있다. 개별 응용 프로그램 별로 성능을 분석해보면 성능 향상의 정도에 따라 다음과 같이 구분할 수 있다.

- **제안 기법의 성능이 LRU와 유사한 경우** : lindsay의 경우에는 제안 기법이 LRU 기법보다 좋은 성능을 보이긴 하지만 그 차이가 거의 미미함을 알 수 있다. 이 경우에는 사실 MRU 기법을 제외한 나머지 기법들이 거의 대부분 유사한 성능을 보이고 있기도 한데, 이는 lindsay가 사용하는 working set이 시간에 따라 거의 변하지 않기 때문으로 판단된다. 시간에 따라 working set이 변하지 않으면 당연히 페이지 폴트가 거의 많이 발생하지 않게

된다. (그림 1)에서 lindsay의 메모리 참조 패턴을 보면 이를 확인할 수 있다.

- **프레임 수에 관계없이 LRU가 항상 좋은 성능을 보이는 경우** : p2c와 grobner, 그리고 gcc에서는 모든 프레임에서 LRU 기법이 좋은 성능을 보인다. 이는 이들 응용의 경우 LFU 정책이 LRU의 성능을 보충해주지 못하고 오히려 해를 입혔다고 할 수 있다. 이도 역시 (그림 1)에 있는 이들 응용들의 메모리 참조 패턴을 분석하면 공통점을 찾을 수 있는데, 이들 응용의 경우에는 한번 참조된 페이지가 시간이 흐름에 따라 계속 반복적으로 참조되었다는 것을 의미한다. (그림 1)에서 이들 세 응용의 경우 중간 중간에 선이 끊긴 경우가 다른 응용에 비해 그리 많지 않은 것으로 확인할 수 있다. 따라서 대부분의 working set이 LRU에 의해 유지될 수 있는 상황에서 LFU 리스트를 유지하기 위해 페이지의 일부를 빼앗김으로써 오히려 제안 기법이 순수 LRU에 비해 좋지 않은 성능을 보인 것으로 판단된다.
- **프레임 수에 따라 제안 기법의 성능이 LRU보다 좋은 경우** : gs와 espresso, gnuplot, 그리고 rscheme에서는 프레임 수에 따라 제안 기법이 LRU보다 좋은 성능을 보이고 있다. gs의 경우 프레임 크기가 350과 500일 때 제안 기법이 LRU 기법보다 각각 0.7%, 2.3% 의 성능 향상을 보인다. espresso의 경우에는 프레임의 크기 50, 60, 70에서 제안 기법이 LRU 기법보다 최대 11.3%의 성능 향상을 보이고 있다. gnuplot에서는 프레임 크기가 5000일 때 제안 기법이 LRU보다 6.8%의 성능 향상을, 5500과 7500에서는 LFU보다 각각 2.4%와 5.1%의 성능 향상을 보이고 있다. 그리고 gnuplot의 프레임 크기 6000과 6500, 7000에서는 LFU 기법이 가장 좋은 성능을 보인다. rscheme의 경우에는 프레임 크기 1400~2000 사이에서 제안 기법이 LRU보다 최대 20.4%의 성능 향상을 보인다. 이들 응용에서 제안 기법이 좋은 성능을 보이는 것도 역시 이들의 메모리 참조 패턴을 보면 알 수 있는데, 이들의 공통점은 그림 중간 중간에 선이 끊기는 경우가 상대적으로 다른 응용에 비해 많다는 점이다. (그림 1)에서 수평선이 끊긴 부분은 "이전에 참조되던 페이지가 당분간 참조되지 않다가 나중에 다시 참조되었다는 것"을 의미한다. LRU만을 사용하고 있는 상황이라면 이 끊긴 기간 동안에 그 페이지는 교체될 가능성이 높다. 하지만 제안 기법에서는 이 페이지가 LRU에 의해 교체 페이지로 선정된다 하더라도 LFU 리스트에 다시 넣기 때문에 이전에 여러 번 참조된 적이 있는 페이지였다면 메모리에 계속 남아 있을 수 있다. 이런 이유로 인해 이들 응용의 경우에는 LRU에 비해 좋은 성능을 보일 때가 있는 것으로 판단된다.

결과적으로 위의 세 가지 경우를 분석해 볼 때 제안 기법은, 과거에 자주 참조했던 페이지를 일정 시간이 경과한 후에 다시 참조하는 패턴을 보이는 응용의 경우에 기존 기법들에 비해 좋은 성능을 보인다고 할 수 있으며, 성능 향상의 정도는 응용 프로그램에 따라 0.7%~20.4%의 페이지 폴트 감소율을 보인다는 것도 알 수 있다.

5.2 리스트의 크기 비율에 따른 실험

5.1절에서 제시한 실험 결과는 제안 기법에서 LRU 리스트의 비율을 5/6로 고정했을 때의 결과이다. 이 비율이 변함에 따라 제안 기법의 성능이 어떻게 변화하는지를 살펴보기 위해 제안 기법의 LRU 리스트의 크기 비율을 5/6, 3/4, 1/2, 1/4, 1/6로 변화시켜 가면서 실험을 하였다. <표 3>은 이 실험 결과를 보이고 있는데, 이 결과도 역시 LRU가 제안 기법

〈표 3〉 리스트 비율에 따른 성능 평가 결과
(각 항목은 페이지 폴트 수. 1위 항목을 음영처리 했음.)

p2c - 전체 참조 횟수 : 30,722,431

프레임 수 / lru 비율	50	60	70	80	90	100	110	120	132
5/6								1037	132
3/4	66648	40379	26976	16731	10230	5314	2661		132
1/2	107170	63525	39787	25081	14668	7437	3588	1199	132
1/4	226384	136659	91297	51863	27998	13974	6267	1522	132
1/6	339928	216208	151320	80287	38863	20201	8395	1846	132

lindsay - 전체 참조 횟수 : 123,690,749

프레임수 / lru 비율	150	200	300	400	450	500	521
5/6	633	632	632	632	522	522	521
3/4	632	632	632	632	522	522	521
1/2	635	635	632	632	522	522	521
1/4	635	635	635		522	522	521
1/6	635	635	635		522	522	521

gs - 전체 참조 횟수 : 134,371,942

lru 비율 \ 프레임 수	100	200	250	300	350	400	500	558
5/6				1145	954	714	576	558
3/4	17591	2659	1476		929	725		558
1/2	27546	5617	2727	1637			616	558
1/4	76597	9066	4770	2429	1229	791	682	558
1/6	125749	11858	5451	2730	1325	795	683	558

grobner - 전체 참조 횟수 : 7,787,835

lru 비율 \ 프레임 수	35	40	45	50	55	60	67
5/6							67
3/4	8103		735	298	108	75	67
1/2	9719	5882	3871	1625	225	78	67
1/4	14907	9978	6687	4411	1010	80	67
1/6	21069	13640	8459	5034	2618	81	67

gcc - 전체 참조 횟수 : 37,524,334

lru 비율 \ 프레임 수	100	200	250	300	350	400	458
5/6							458
3/4	37621	8091	3743	1885	1114	721	458
1/2	53052	10350	5366	3393	2045	924	458
1/4	89544	16488	9218	5251	3151	1669	458
1/6	113393	21248	11354	6244	3626	1749	458

espresso - 전체 참조 횟수 : 326,938,361

lru 비율 \ 프레임 수	20	30	40	50	60	70	77
5/6				162		78	77
3/4		4955		163		78	77
1/2	123231	8508	1729		92	78	77
1/4	261754	55475	10357	1215	108	78	77
1/6	707884	96467	29514	2165	291	78	77

gnuplot - 전체 참조 횟수 : 68,458,509

lru 비율 \ 프레임 수	5000	5500	6000	6500	7000	7500	7718
5/6	21555	21368	21167	19443	18992	12799	7718
3/4	21555	21368	19892	19443	18992	12799	7718
1/2			19892			12799	7718
1/4			19892			12799	7718
1/6						12799	7718

rscheme - 전체 참조 횟수 : 151,606,733

lru 비율 \ 프레임 수	1000	1200	1400	1600	1800	2000	2039
5/6				6469	3547	2107	2039
3/4	22584	15849	10718	6433	3294	2103	2039
1/2	29237	18644	11573				2039
1/4	41956	25985	15341	7316	3281	2093	2039
1/6	51254	31021	17431	8064	3372	2095	2039

보다 나은 성능을 보이는 응용과 성능이 유사한 응용, 그리고 제안 기법에서 좋은 성능을 보일 때가 있는 응용에 따라 조금씩 차이를 보이고 있다.

LRU가 제안 기법보다 항상 좋은 성능을 보였던 p2c와

grobner, gcc의 경우에는 LRU 리스트의 크기 비율이 커질수록 프레임 개수에 상관없이 제안 기법은 좋은 성능을 보여준다. 이들 응용에서는 LRU가 가장 좋은 성능을 보이고 있기 때문에 당연한 결과라 하겠다. 한편 제안 기법과 LRU가 유사한 성능을 보였던 lindsay에서는 프레임 크기가 400일 때를 제외하면 LRU 리스트 크기 비율이 성능에 별다른 영향을 주지 못하고 있음을 알 수 있다. 반면, 제안 기법이 LRU에 비해 좋은 성능을 보일 때가 있었던 gs, espresso, gnuplot, rscheme에서는 LRU 리스트 크기가 커진다고 해서 항상 성능이 좋아지지는 않는다는 것을 알 수 있다. gs의 경우, 프레임 크기 100, 200, 250에서는 LRU 리스트의 크기 비율이 커질수록 성능이 좋아지지만 그 외의 프레임 크기에서는 그렇지 않다. espresso의 경우에는 프레임 크기 20, 40, 60에서 LRU 리스트의 크기 비율이 5/6과 3/4일 때 같은 성능을 보이고, 프레임 크기 50에서는 1/2일 때 오히려 가장 좋은 성능을 보이고 있다. gnuplot에서는 LRU 리스트의 크기 비율이 1/2 이하일 때가 보다 좋은 성능을 보인다. rscheme의 경우에도 프레임 크기가 1000, 1200, 1400일 때에만 LRU 리스트의 크기 비율이 커질수록 성능이 좋아질 뿐 나머지 경우에는 그렇지 않다.

위의 결과에서 알 수 있는 것은 LRU 리스트의 크기가 커진다고 해서 페이지 폴트의 횟수가 모든 응용에서 감소하지는 않는다는 점이다. 바로 이러한 점 때문에 LRU 기법과 LFU 기법을 동시에 적용하는 제안기법이 LRU를 능가하는 성능을 보일 때가 있는 것으로 판단된다.

6. 결 론

8개의 최초 응용 트레이스에 대해서 참조 패턴을 분석하면 일부 응용을 제외한 대부분의 응용에서 참조되는 페이지는 최근성과 참조 횟수 모두에 의해 가치가 결정된다. 다시 말해서 어떤 경우에는 최근에 참조된 페이지가 계속 참조되며, 또 다른 경우에는 자주 참조되는 페이지가 계속 참조되는 경향이 있다. 분석 결과 대부분의 응용에서 참조되는 페이지의 최근성과 참조 횟수 모두가 페이지의 가치를 결정하는 중요한 요소라는 것을 확인할 수 있었다. 따라서 최근성만을 가지고 페이지를 교체하는 순수 LRU 기법이나, 참조 횟수만을 가지고 페이지를 교체하는 순수 LFU 기법만으로는 페이지 교체를 최적화시키기 어렵다. 참조의 최근성과 참조 횟수 모두를 고려하는 제안 기법은, 과거에 자주 참조했던 페이지를 일정 시간이 경과한 후에 다시 참조하는 패턴을 보이는 응용의 경우에 기존 기법들에 비해 좋은 성능을 보인다. 성능 향상의 정도는 응용 프로그램에 따라 0.7%~20.4% 정도라는 것을 실험을 통해 확인할 수 있었다.

제안 기법에서 LRU 리스트의 크기 비율을 다양하게 주어 실험해 본 결과 제안 기법에 비해 LRU가 나은 성능을 보이는 응용에서는 LRU 리스트 크기 비율이 커질수록 좋은 성능을 보인다는 것을 알 수 있었다. 하지만 LRU 리스트 비율과 성능이 비례하지 않는 응용도 있다는 점을 아울러 발견했으며, 이로 인해 본 논문에서 제시한 기법이 LRU를 능가하는 성능을 보일 때가 있는 것으로 판단된다. 극단적인 예로 반복적인 패턴을 갖는 gnuplot 응용의 경우에는 MRU 기법이 좋은 성능을 보이기 때문에 LRU 리스트의 비율을 아무리 늘려도 성능에는 영향을 미치지 못하며, LRU와 LFU를 동시에 고려하는 본 제안기법이 LRU에 비해 좋은 성능을 나타냈다.

향후 연구과제로 프로세스의 코드를 세그먼트 별로 나누어서 각 세그먼트 별로 참조 패턴을 분석하고, 참조 패턴에 따라 동적으로 교체 기법을 달리 적용하는 기법을 연구할 계획이다. 또한, 페이지 리스크를 LRU와 LFU로 분리하지 않으면서도 LRU와 LFU의 장점을 동시에 활용할 수 있는 페이지 가치도에 기반한 페이지 교체 기법에 대해서도 연구할 계획이다.

참 고 문 헌

[1] 이동희, "LRFU 블록 교체 기법", 서울대학교 컴퓨터공학과 박사학위 논문, 1998.

[2] 조유근, 고건, *운영체제론*, 홍릉과학출판사, 1990.

[3] 최종무, 조성제, 노삼혁, 민상렬, 조유근, "적응력있는 블록 교체 기법을 위한 효율적인 버퍼 할당 정책", 정보과학회논문지 : 시스템 및 이론, 제27권 제3호, 2000.

[4] Bach, M. J., *The Design of the Unix Operating System*, Englewood Cliffs, N. J. : Prentice-Hall, 1986.

[5] Denning, P. J., "The Working Set Model for Program Behavior," CACM, Vol.11, No.5, pp.323-333, May, 1968.

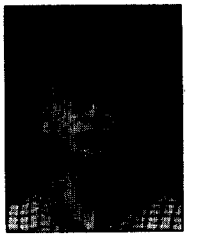
[6] G. Glass, P. Cao, "Adaptive Page Replacement Based on Memory Reference Behavior," In Proceedings of the 1997 ACM SIGMETRICS Conference, pp.115-126, June, 1997.

[7] Rusling D. A., *The Linux Kernel*, Linux Documentation Project, 1999.

[8] Silberschatz A., Galvin P., *Operating System Concepts*, Addison-Wesley, 1998.

[9] Tanenbaum, A. S., *Mordern Operating Systems*, Englewood Cliffs, N. J., Prentice-Hall, 1992.

[10] Y. Smaragdakis, S. Kaplan, and P. Wilson, "EELRU : Simple and Effective Adaptive Page Replacement," in SIGMETRICS '99.



이 승 훈

e-mail : zor4@dankook.ac.kr

1998년 단국대학교 전산통계학과(학사)

2001년 단국대학교 전산통계학과(석사)

2001년~2001년 WinsRoad(주)

관심분야 : 운영체제

이 종 우

e-mail : jwlee44@hallym.ac.kr

1990년 서울대 컴퓨터공학과 졸업(학사)

1992년 서울대 컴퓨터공학과 석사과정 졸업(석사)

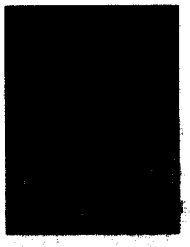
1996년 서울대 컴퓨터공학과 박사과정 졸업(박사)

1996년~1998년 현대전자산업(주) 과장

1998년~1999년 현대정보기술(주) 책임연구원

1999년~현재 한림대학교 정보통신공학부 조교수

관심분야 : 운영체제, 병렬/분산 운영체제, 클러스터 시스템, 전산금융



조 성 제

e-mail : sjcho@dankook.ac.kr

1989년 서울대학교 전자계산기공학과 졸업

1991년 서울대학교 컴퓨터공학과 석사

1996년 서울대학교 컴퓨터공학과 박사

1996년~1997년 서울대학교 컴퓨터신기술 연구소 연구원

1997년~현재 단국대학교 전산통계학과 조교수.

관심분야 : 운영체제, 시스템소프트웨어, 분산공유메모리 아키텍처