효율적인 버퍼 캐시 관리를 위한 동적 캐시분할 블록교체 기법

진재선*, 허의남**, 추현승***

Dynamic Cache Partitioning Strategy for Efficient Buffer Cache Management

Jaesun Jin, Eui-Nam Huh, Hyunseung Choo

Abstract

The effectiveness of buffer cache replacement algorithms is critical to the performance of I/O systems. In this paper, we propose the degree of inter-reference gap (DIG) based block replacement scheme that retains merits of the least recently used (LRU) such as simple implementation and good cache hit ratio (CHR) for general patterns of references, and improves CHR further. In the proposed scheme, cache blocks with low DIGs are distinguished from blocks with high DIGs and the replacement block is selected among high DIGs blocks as done in the low inter-reference recency set (LIRS) scheme. Thus, by having the effect of the partitioning the cache memory dynamically based on DIGs, CHR is improved. Trace-driven simulation is employed to verified the superiority of the DIG based scheme and shows that the performance improves up to about 175% compared to the LRU scheme and 3% compared to the LIRS scheme for the same traces.

Key Words: Buffer Cache Management, Buffer Replacement Policy, Trace-Driven Simulation

* 성균관대학교 정보통신공학부
** 서울여자대학교 컴퓨터공학과 교수
*** 성균관대학교 정보통신공학부 교수
책임저자 : 추현승

1. 서론

안정적인 I/O 관리는 시스템 전체의 성능에 큰 영향을 미치므로, 버퍼 캐시 관리를 위한 관리 기법들에 관한 연구는 오래 전부터 이루어져 왔다. 버퍼 캐시 관리 기법은 저장장치의 계층적 구조로 인해 발생하는 캐시 적중 실패를 효과적으로 관리할 수 있어야 하며, 저장장치의 빈번한 접근으로 인한 관리 기법의 오버헤드를 최소화해야 한다. 그러나 효율성과 단순성을 동시에 만족시키기 어렵다. 효율적인 버퍼 캐시 관리를 위해 다양한 방법들이 연구되고 추가되었지만, 이로 인해 발생하는 오버헤드는 관리 기법의 성능 저하를 가져온다.

버퍼 캐시 관리의 효율성과 단순성을 동시에 만족시키기 위해 많은 기법들이 제안되고 있으나, 지금까지는 일반적인 참조 패턴에 대해 좋은 관리 능력을 보이면서 관리 기법의 오버헤드를 최소로 한 LRU(Least Recently Used)가 가장 많이 사용되고 있다. 그러나 LRU의 버퍼 캐시 관리 효율은 기법의 단순성에 비해 효과적이지 못하다. 최근 활발하게 연구되고 있는 탐지 기반의 기법들은 탐지를 위한 메모리 및 시간적 오버헤드가 크다. 따라서 본 논문에서는 현재까지 가장 효과적이라고 알려진 LRU의 장점인 단순성(simplicity)을 유지하면서 참조 패턴을 탐지 하는 블록 교체 기법을 제안한다. 본 논문에서 제안하는 동적 캐시 분할 블록 교체 기법은 LIRS[6] 기법에 그 근본을 두고 있으며, LIRS의 문제점을 파악하고 이를 해결하는데 초점을 맞추도록 한다. 또한, 지금까지의 연구는 관리하는 버퍼의 크기에 대한 고찰이 없다. 그러나 이동형 장비의 프로세서 능력이 향상되고, 메모리 및 저장장치의 용량이 증가함에 따라, 제한된 버퍼 크기에서의 효율적인 버퍼 관리 기법이 요구된다. 그래서 본 논문에서는 제한된 버퍼 크기에서의 성능을 평가하여 이동형 장치에도 적용될 수 있는 블록 교체 기법을 제안한다.

본 논문의 구성은 다음과 같다. 제 2절에서는 기존에 연구된 블록 교체 기법들에 대한 설명과 문제점을 제시하고, 제 3절에서는 본 논문에서 제안하는 동적 캐시 분할 블록 교체 기법을 설명하고, 제 4절에서는 시뮬레이션 결과를 확인하며, 본 논문의 결론을 서술한다.

2. 버퍼 캐시 관리 기법

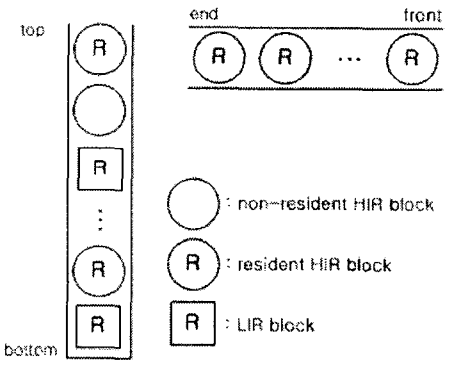
지금까지 효율적인 버퍼 캐시 관리를 위해 제안된 기법들로는 2Q [4], SEQ [7], EELRU (Early Eviction Least Recently Used) [8], LRFU (Least Recently/Frequency Used) [5], UBM (Unified Buffer Management) [10], LIRS 등 많은 기법들이 존재하며, LRU보다 더 많은 정보를 이용하거나, 참조 패턴을 검출하여 이에 따라 교체 방법을 선택하는 방법을 이용함으로써 LRU의 문제점을 해결하였다. 이처럼 LRU를 기반으로 버퍼나 페이지 캐시를 관리하기 위해 교체시킬 버퍼나 페이지를 선택하는 많은 기법들이 존재하며, 이들 기법들은 크게 세 가지로 분류된다. 첫 번째는 사용자의 힌트를 기반으로 하는 기법 (User-level hint)이고, 두 번째는 역사 정보에 기반을 두는 기법 (Tracing and utilizing history information)이며, 세 번째는 탐지 기반의 기법 (Detection and Adaptation of Access Regularities)이다. 본 절에서는 세 가지 분류와 각 분류에 속하는 기법들을 소개한다.

2.1 사용자의 힌트 기반 기법

사용자의 힌트를 기반으로 하는 기법들은 사용자가 제공하는 힌트를 기초로 하여 가까운 미래에 사용될 확률이 적은 블록을 선정하는 방식으로, ACFC 기법과 AIPC 기법이 이에 해당된다. 적절한 힌트를 제공하기 위해서 사용자는 데이터 접근 패턴에 대한 충분한 이해를 가지고 있어야 한다.

2.2 역사 정보 기반 기법

역사 정보들을 기반으로 하는 기법들은 각 블록들이 과거에 참조되었던 시간 정보나 참조 사이의 시간 간격 등의 정보들을 저장하고 이를 활용하여 희생될 블록을 선택하는 방법이다. 이러한 기법들은 일반적으로 메모리와 시간적인 오버헤드가 적고 구현이 간단하다는 장점을 가지지만, 상대적으로 탐지 기반의 기법들에 비해 캐시 적중률이 낮다. [10] 역사 정보를 기반으로 하는 기법들에는 LRU, LRU-K, 2Q, LRFU, LIRS와 같은 기법들이 이에 해당된다.



<그림 1> LIRS 버퍼 교체 기법

역사 정보 기반 기법에서 유일하게 탐지 기반 기법보다 캐시 적중률이 높은[6] LIRS 버퍼 교체 기법은 한 블록에 대한 참조들 간의 최근성(inter-reference recency)을 고려하여 블록들을 관리하는 기법이다. 여기서는 블록들은 LIR (Low Inter-reference Recency) 블록과 HIR (High Inter-reference Recency) 블록으로 구분한다. 그리고 전체 블록들을 참조된 순서에 의해 LRU 리스트로 유지하며, 추가로 교체를 위한 블록들인 HIR 블록들을 또 다른 리스트로 관리한다. LIRS에서는 상대적으로 오래 전에 참조된 블록은 이후에도 오랜 시간 후에 참조될 것이라는 가정 하에 상호 참조 간격이 큰 HIR 블록을 교체하며, LIR 블록은 교체 대상에서 제외된다. 그리고 HIR 블록의 참조 간격이 작아지면 리스트 상의 가장 오래된 LIR 블록과 상태 변화를 일으켜, HIR 블록은 LIR 블록이 되고, LIR 블록은 HIR 블록으로 만든다. 이렇게 함으로써 자주 참조될 확률이 큰 블록(LIR 블록)들을 계속 캐시에 유지함으로써 캐시 적중률을 개선한 방식이다.[6] 그러나 이 기법에서는 HIR 블록들과 LIR블록들이 관리되는 리스트의 크기를 정적으로 정해주어야 하는 단점이 있다.

2.3 탐지 기반 기법

탐지 기반의 기법들은 각각의 참조에 대해 참조 패턴을 검출하고 이에 따라 분류하여 각각의 분류된 참조들에 대해 다른 기법을 적용하여 관리하는 기법이다. 참조 패턴을 탐지함으로써 각 참조들에 대해 적응적으로 관리할 수 있어서 캐시 적중률이 상대적으로 높으며, 최근에 가장 많이 연구되고 있는 방식이다. 하지만 분리하여 관리하기 위한 메모리와 시간적인 오버헤드가 크다는 단점이 있다.[6] 이러한 탐지 기반 기법에 속하는 기법들로는 SEQ, EELRU, DEAR, UBM 등이 있다.

UBM 버퍼 교체 기법은 검출 테이블에 기반을 두어 참조되는 블록들을 순차 참조와 순환 참조, 그리고 기타 참조로 분류하고, 분류된 참조들에 대해서 서로 다른 기법을 적용한다. 캐시 접근 실패가 발생하면 교체할 블록을 선정하기 위해 한계 효용 값을 계산하고 이를 토대로 세 가지 캐시 공간 중에서 적절한 공간을 선택한 뒤에, 그 공간에 적용되는 교체 기법을 통해 블록을 교체한다. 하지만 이 기법은 테이블을 유지해야 하며 캐시 공간들을 분리해서 관리해야 하는 오버헤드가 필요하다.[10]

3. 동적 캐시 분할 블록 교체 기법

3.1 기본 개념

동적 캐시 분할 기법은 역사 정보 기반 기법의

LIRS 블록 교체 기법에 기반을 두고 있다. LIRS 블록 교체 기법은 2절에서 설명되었듯이 상대적으로 참조 간격이 긴 블록은 한번 참조 된 후 다시 오랜 간격 후에 참조된다고 가정한다. 그래서 캐시를 참조 간격이 긴 블록들의 집합과 짧은 블록들의 집합으로 분리하여 관리한다. 캐시 적중 실패가 발생하면 참조 간격이 긴 블록들이 교체 대상으로 선정된다. 그래서 캐시 적중 실패를 줄이게 된다.

동적 캐시 분할 기법은 캐시를 HIG (High Inter-reference Gap)와 LIG (Low Inter-reference Gap)으로 분류한다. LIG 블록은 자주 참조되는 블록이므로 계속 캐시에 유지하고, HIG 블록 중에서 교체 대상을 선정한다. 캐시에 저장된 블록들은 HIG, LIG 중의 하나의 상태를 유지하게 된다. 블록의 상태 결정은 블록이 캐시에 올라올 때 결정되며, 캐시에서 관리되는 과정에서 블록의 상태가 변경될 수 있다. 또, HIG 블록은 HIG와 nHIG 두 블록상태로 구분될 수 있다. HIG 블록 중에서 현재 캐시에 존재하고 있는 상태는 HIG상태이며, nHIG 블록은 캐시를 관리하기 위한 자료구조만 유지되고 있는 상태를 나타낸다. 그러므로 캐시에 존재 하는 블록은 LIG, HIG, nHIG 의 세 가지 상태 중에 하나이며, 동시에 두 가지 이상의 상태로 존재하거나, 상태가 없이 캐시에 존재할 수 없다.

전체 캐시의 크기를 L이라 하고, HIG의 블록 크기를 L_{HIG}, LIG의 블록크기를 L_{LIG} 이라 하면 $L=L_{HIG}, + L_{LIG}$ 이 된다. L_{HIG}의 크기와 L_{LIG}의 크기는 참조되는 패턴에 의해 동적으로 변하게 된다. 본 기법은 역사 정보 기반 교체 기법에 탐지 기반 기법을 추가하여, 일정한 패턴을 가지고 참조되는 블록에 대해서 가변적으로 버퍼 크기를 조정하여 캐시 적중률을 높이기 위한 방법을 제시한다.

동적 캐시 분할 기법에서 블록이 관리되는 과정은 다음과 같다. 캐시에서 블록이 참조되는 경우는 (1) LIG 블록이 참조되는 경우, (2) HIG블록이 참조되는 경우, (3) nHIG블록이 참조되는 경우와 (4) 블록이 처음으로 참조되는 경우가 있다. (3)과 (4)는 캐시에 존재하지 않는 블록이 참조된 경우로 본 기법에서는 처리 과정이 같으므로 특별히 분류해서 설명하지 않는다.

(1) LIG 블록이 참조된 경우에는 캐시 적중 성공 상태이다. 이미 캐시 상에 존재하고 있는 블록에 대한 접근이므로, 블록을 관리하고 있는 자료의 갱신만이 일어난다. (2) HIG블록이 참조되는 경우에는 두 가지로 나누어 생각할 수 있다. 첫째, 연속된 참조 패턴으로 직전의 참조가 LIG 블록이었을 경우에는 참조된 블록을 LIG블록으로 상태 변화를 한다. 그러므로 전체적으로 볼 때 LIG블록은 한 개 추가 되며, HIG블록은 한 개 감소하게 된다. 둘째, 참조 패턴에 의한 참조가 아닐 경우에는 블록을 관리 하고 있는 자료를 갱신하고, LIG블록 중에서 가장 참조가 오래된 블록과 비교하여, 지금 참조된 블록의 참조 거리가 짧을 경우에는 두 블록의 상태를 서로 교환한다. 그러므로 전체적으로 LIG 블록과 HIG블록의 개수는 변화가 없게 된다. (1)과 (2) 의 경우는 캐시 적중 상태이므로 희생되는 블록이 발생하지 않는다. (3) nHIG블록이 참조되는 경우와 (4) 블록이 처음 참조되는 경우에는 블록이 현재 캐시에 존재하지 않는 경우이다. 이러한 경우에는 HIG블록 중에서 가장 참조 거리가 긴 블록이 희생되고, 새로운 블록이 HIG블록으로 등록된다.

<표 1>은 동적 캐시 분할 기법이 어떻게 블록을 교체하고, 동적으로 HIG와 LIG 블록을 관리하는 지를 보여준다. 표 1에서 'X'는 단위 가상 시간 n에서 참조 되고 있는 블록을 나타낸다. 즉, 블록 A는 단위 가상 시간 1과 7에서 참조되고 있음을 나타낸다. 최근성을 나타내는 R (Recency)은 가장 최근에 참조된 순서를 나타내며, rIG (relative Inter-reference Gap)은 상대적인 블록 참조 거리를 나타낸다. 상대적인 블록 참조 거리 rIG는 블록이 참조된 후 다시 참조 될

때까지의 거리를 상대적으로 계산한 값이다. 연속해서 참조되는 블록 중에서 같은 rIG를 갖는 블록은 이전 블록과 함께 참조되는 블록이므로, 이전 블록이 HIG에 속하고, 현재 참조되는 블록이 LIG에 속해 있으면, 현재 블록을 LIG블록에서 HIG 블록으로 상태 변화를 시킨다.

<표 1> 동적 캐시 분할 기법, $L=L_{HIG} + L_{LIG} = 2+1$, 참조패턴={A,D,B,C,E,D,A,B,C,D,B,C}

time	A	B	C	D	E	LIG/HIG
1	X					1/2
2				X		1/2
3		X				1/2
4			X			1/2
5					X	1/2
6				X		1/2
7	X					1/2
8		X				1/2
9			X			**2/1**
10				X		**2/1**
11		X				**2/1**
12			X			**2/1**
R	3	1	0	2	4	
rIG	5	2	2	3	∞	

상대 참조 거리를 계산하는 이유는 알고리즘의 시간적, 공간적 복잡도를 증가시키지 않고 참조 패턴을 분석하기 위해 도입된 방법이며, 다음절에서 구현 방법을 자세히 설명한다. 그러므로, 단위 가상 시간 12일 때, 블록 E, D, C, B, A에서 최근성은 4, 2, 0, 1, 3이고, rIG는 무한대(∞), 3, 2, 2, 5가 된다. LIG/HIG는 단위 가상 시간에서의 LIG/HIG의 비율을 나타내며, 버퍼 캐시의 크기 L=3이다.

<표 2> LIRS와 동적 캐시 분할 블록 교체 기법의 비교, (*)는 블록 적중 실패

time	C_1	C_2	C_3	C_1	C_2	C_3
1	**A**$^{(*)}$			**A**$^{(*)}$		
2	**A**	D$^{(*)}$		**A**	D$^{(*)}$	
3	**A**	D	B$^{(*)}$	**A**	D	B$^{(*)}$
4	**A**	B	C$^{(*)}$	**A**	B	C$^{(*)}$
5	**A**	C	E$^{(*)}$	**A**	C	E$^{(*)}$
6	**A**	E	D$^{(*)}$	**A**	E	D$^{(*)}$
7	**A**	E	D	**A**	E	D
8	**A**	D	B$^{(*)}$	**A**	D	B$^{(*)}$
9	**A**	B	C$^{(*)}$	**C**$^{(*)}$	**B**	A
10	**A**	C	D$^{(*)}$	**C**	**B**	D$^{(*)}$
11	**A**	D	B$^{(*)}$	**C**	**B**	D
12	**A**	D	C$^{(*)}$	**C**	**B**	D
	LIRS			동적캐시분할기법		

<표 2>에서는 LIRS와 동적 캐시 분할 블록 교체 기법이 각각 버퍼 캐시를 어떻게 관리하는지를 보여준다. 동적 캐시 분할 블록 교체 기법에서 단위 가상 시간 9에서 블록 C와 B가 같은 패턴으로 참조되므로, 블록 C가 LIG로 상태 변화를 한다. 즉, 단위 가상 시간이 8일 때, LIG={B}, HIG={A, D}이고, 단위 가상 시간 9에서 LIG={B, C}, HIG={A}로 변하게 된다. 동적 캐시 분할 블록 교체 기법을 LIRS와 비교해 볼 때, 일정한 패턴을 가지고 자주 참조되는 블록들을 LIG의 확장을 통해 캐시에 머무르게 하여 캐시 적중 실패를 줄이는 것을 <표 2>의 단위 가상 시간 9 이후에서 볼 수 있다. <표 1>과 같은 참조 패턴과 초기 조건일 때, LIRS의 블록 적중 실패는 총 11회 이고, 동적 캐시 분할 블록 교체 기법의 블록적중 실패는 9회로 줄어듦을 확인할 수 있다. 일정 시간 동안 (30초) 증가한 LIG의 블록이 참조 되지 않으면 증가된 부분은 다시 HIG로 반환된다.

3.2 LRU 스택과 rIG를 이용한 동적 캐시 분할 기법의 구현

동적 캐시 분할 기법은 LRU 스택과 HIG블록의 큐를 이용하여 구현했다. LRU 스택에는 캐시 전체가 관리되며, 캐시 적중 실패에 따른 빠른 블록의 할당을 위해 HIG블록을 사이즈가 L_{LIG}를 넘지 않는 큐 형태로 관리한다. LRU 스택을 이용한 이유는 프로그램의 시간적 복잡도를 LRU와 같이 유지하고, IG를 계산하기 위한 추가적으로 메모리 공간을 사용하지 않아도 되는 이점이 있기 때문이다. 블록이 참조되면 참조된 블록은 LRU 기법에 의해서 블록이 LRU 스택의 top에 위치하게 되고, LIG 중에서 IG가 가장 큰 블록이 LRU 스택의 bottom에 위치하게 된다. LRU 스택의 bottom에 항상 LIG 중에서 IG가 가장 큰 블록을 위치하기 위해 스택 정리 기법을 이용했다. 스택 정리 기법이란 참조된 블록이 LRU 스택의 bottom에 위치한 LIG블록이고, LRU 스택 관리 기법에 의해 스택의 top으로 이동한 후, 다음 LIG블록이 나올 때까지 bottom의 블록들을 제거하는 기법을 말한다. 항상 LRU 스택의 bottom에 LIG블록이 존재하게 함으로써 HIG블록과의 상태 변화를 가능하게 한다. LRU 스택의 bottom에 HIG블록이 존재하면 LRU 스택 특성상 스택 bottom의 블록보다 더 큰 HIG 블록이 스택 중간에 존재할 수 없으므로 더 이상 블록의 상태변화와 L_{HIG}와 L_{LIG}의 크기를 동적으로 변화시킬 수 없기 때문이다. LRU 스택을 이용하면 LIG 블록 중에서 IG(Inter-reference Gap)값이 작은 블록들은 계속 LRU 스택의 top으로 이동하게 되므로 IG의 계산 없이 캐시에 계속 남아 있게 된다.

IG는 각 블록마다 이전 참조 후 다음 참조까지의 블록 개수를 의미한다. 그러나 알고리즘의 시간적 복잡도를 고려하여 LRU 스택을 이용하여 구현되었기 때문에 참조되는 각 블록들의 IG 값은 계산될 수 없다. 그래서 블록이 처음 참조된 후 다음 참조까지의 최대 블록 개수를 개산하여 rIG를 계산한다. 각 블록마다 계산된 rIG는 블록의 참조 패턴을 파악할 수 있으며 연속된 블록 참조 중에서 같은 rIG를 갖는 블록은 항상 함께 참조되는 블록으로 분류 될 수 있다. 그러므로 rIG를 계산하면 프로그램의 시간적 복잡도를 증가시키지 않고 패턴을 감지하여 캐시를 유연하게 운영할 수 있다. 특히 제한된 캐시의 크기를 가진 이동형 장비에서는 rIG를 이용한 참조 패턴의 분석과 이를 이용한 동적 캐시 분할 기법의 성능 향상을 확인할 수 있다.

4. 성능평가

본 장에서는 트레이스 기반 시뮬레이션 방법으로 동적 캐시 분할 기법과 각 기법들에서 가장 좋은 성능을 보이고 있는 UBM과 LIRS[6], 그리고 최적(Off-line Optimal)기법과 성능을 비교한다.

4.1 기본 개념

실험에 사용된 트레이스들은 UBM 기법의 저자들이 제공한 FreeBSD 운영체제에서 여러 응용들을 동시에 수행시키면서 수집된 자료를 이용하였다. 각 응용들의 특성은 다음과 같다.

cscope: cscope는 대화형 C-프로그램 검색 도구로 검색 대상이 되는 C-프로그램들로부터 인덱스 파일을 생성한 후, 요청된 C 심볼과 함수를 찾는 도구이다. 수행 시 기타 참조 패턴과 하나의 순환 참조 패턴이 검출된다.

glimpse: glompse는 텍스트 문서 검색 도구로써, 검색 대상이 되는 텍스트 문서들로부터 인덱스 파일을 생성한 후 요청된 단어를 찾는 도구이다. 수행 시 기타 참조 패턴, 순차 참조 패턴, 그리고 서로 다른 순환 주기를 갖는 다수의 순환 참조 패턴들이 검출된다.

gnuplot: gnuplot은 대화형 그래프 작성 도구이다. 수행 시 기타 참조 패턴과 하나의 순환 참조 패턴이 검출된다.

cpp: cpp는 GNU C 언어를 위한 전처리기이다. 수행 시 기타 참조 패턴과 순차 참조 패턴이 검

출된다.
postgres: postgres는 관계형 DB 시스템이다. 수행 시 기타 참조 패턴, 순차 참조 패턴, 그리고 서로 다른 순환 주기를 갖는 다수의 순환 참조 패턴이 검출된다.

<표 3> 동시에 수행된 응용프로그램들의 집합

트레이스	수행된 응용프로그램	참조 횟수	참조된 블록수
Multi1	cscope, cpp	15858	2606
Multi2	cscope, cpp, postgress	26311	5684
Multi3	cpp, gnuplot, glimpse, postgress	30241	7453

<표 3>은 실험에서 사용된 각 트레이스 별로 생성 시 수행된 응용들과 각 응용들이 참조한 블록 수와 총 참조 횟수를 보여준다. 다른 여러 가지 기법들과 시뮬레이션 결과를 비교하기 위해서는 다른 기법들의 실험 인자들을 조정할 필요가 있다. 본 논문에서는 이러한 인자들을 각 논문에서 저자들이 제시한 인자 값들을 이용하였다.

4.2 시뮬레이션 결과

본 논문에서 제안하는 동적 캐시 분할 기법은 LRU 스택을 이용하여 알고리즘의 시간적 복잡도를 LRU와 같이 유지하면서 캐시 적중률을 높이기 위해 고안되었다. 또한 제한적인 캐시 크기를 갖는 시스템에서 캐시를 유동적으로 관리 할 수 있는 방안을 제시하였다. 또한 역사정보 기반의 캐시관리 기법에 특별한 오버헤드의 증가 없이 특정한 참조 패턴을 검출할 수 있게 한 것이 장점이다. 본 절에서는 제한된 캐시 크기에서 다른 여러 가지 블록 교체 기법들과 성능을 비교한다.

<그림 2>와 <표 4>에서 각 트레이스에 따른 캐시 적중률을 보여준다. 이를 통해 제안하는 동적 캐시 분할 기법이 오프라인 최적 알고리즘에 가까운 캐시 적중을 가짐을 알 수 있다. 제한된 캐시 크기에서의 적중률을 보기 위해 최대 캐시 블록의 개수를 1,000개로 제한했다. 시뮬레이션에서 사용된 3개의 트레이스 (Multi1, Multi2, Multi3) 모두에서 LIRS에 비해 평균 3%의 성능 향상을 이루었으며 UBM 기법에 비해서는 평균 175%까지 성능 향상을 이루었음을 볼 수 있다.

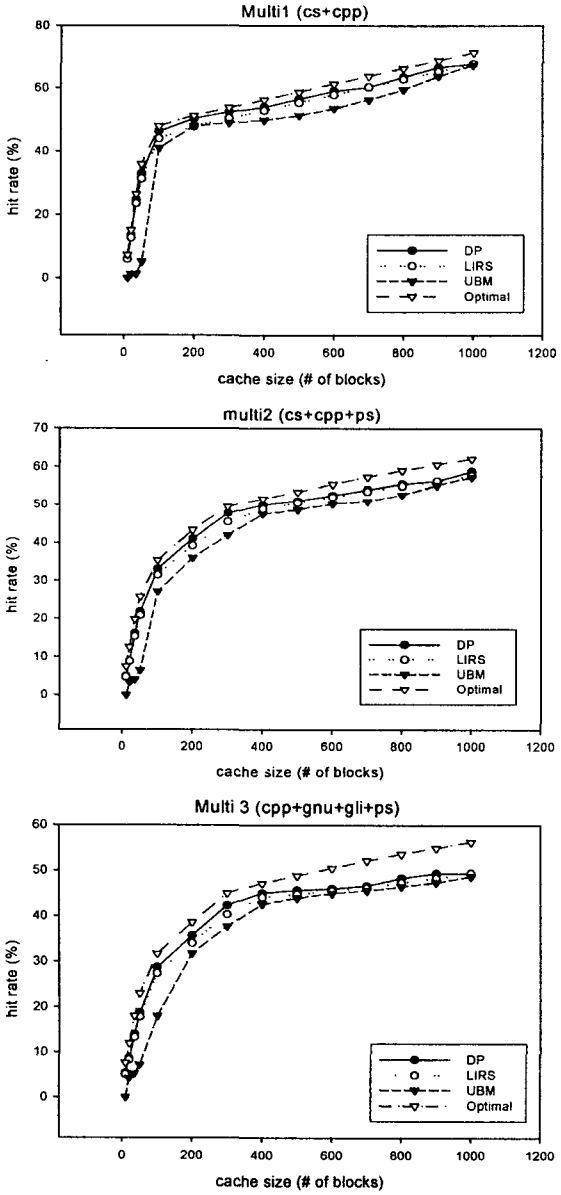
성능평가를 위해 비교한 3가지의 버퍼 캐시 관리 알고리즘은 모두 오프라인 최적 알고리즘에 가까운 캐시 적중률을 보인다. 하지만 UBM 기법은 참조 패턴을 검출해야 하는 시간적인 오버헤드와 테이블 관리를 위한 추가적인 메모리 오버헤드가 필요하다. 본 논문의 기초가 되는 LIRS 블록 교체 기법과 본 논문에서 제안하는 동적 캐시 분할 기법의 성능은 결과로 확인할 수 있듯이 거의 유사하다. 하지만 제한된 캐시 크기 (1,000개 이하)에서는 전체적으로 LIRS 보다 좋은 성능을 보임을 알 수 있다. 이는 제한된 크기 내에서 rIG를 이용한 참조 패턴 검출 기법을 이용하여 유동적으로 HIG와 LIG를 관리함으로써 일정한 패턴을 가지고 참조되는 접근을 오랫동안 LIG에 머무르게 하여 캐시 적중률을 높이기 때문이다.

5. 결론

효율적인 버퍼 캐시 관리는 알고리즘의 단순성과 관리의 효율성을 모두 해결해야 하는 문제이다. 본 논문에서는 현재 널리 쓰이고 있는 LRU 기법의 단순성을 유지하면서, 참조 패턴의 분석을 추가적인 오버헤드 없이 검출하여 동적으로 캐시를 관리하는 기법을 제시했다. 또한, 제한된 캐시 크기에서의 관리의 효율성을 입증함으로써 이동 컴퓨팅 환경의 변화에 적절하게 대응할 수 있는 버퍼 캐시 관리 기법을 보였다. 성능 평가에서 확인할 수 있듯이 동적 캐시 분할 블록 교체 기법은 LIRS에 비해 평균 3%, UBM에 비해 평균 175%까지의 성능 향상이 있었음을 보여준다.

향후 연구로 동적 캐시 분할 블록 교체 기법을

FreeBSD나 Linux 운영체제에 적용하여 실제로 어느 정도의 성능 향상을 보이는지 측정하게 될 것이다. 또한 본 논문에서 제안한 것과 같이 시간적, 공간적 복잡도를 증가시키지 않으면서 특정한 패턴을 검출 할 수 있는 기법에 대해 연구를 계속해 볼 예정이다.



<그림 2> Multi 1, 2, 3 트레이스 결과

<표 4> Multi 1, 2, 3에서의 캐시 적중률

Multi 1			
# of blocks	LIRS	DP	UBM
20	12.73	13.37	1.19
35	23.48	24.65	1.27
50	31.32	32.89	5.15
100	43.98	46.26	41.01
200	47.84	50.22	47.94
300	50.40	52.41	48.84
400	52.88	53.93	49.80
500	55.34	56.45	51.29
600	57.86	59.00	53.53
700	60.27	60.27	56.27
800	62.83	63.46	59.48
900	65.25	66.56	63.76
1000	67.73	67.73	67.37
Multi 2			
# of blocks	LIRS	DP	UBM
20	8.85	8.78	3.34
35	15.34	16.11	4.10
50	20.79	21.83	6.43
100	31.49	33.06	27.06
200	39.13	41.09	35.97
300	45.43	47.70	42.01
400	48.94	49.92	47.52
500	50.58	51.09	48.76
600	52.13	52.65	50.29
700	53.67	54.20	51.04
800	55.06	55.61	52.85
900	56.34	56.34	55.32
1000	57.73	58.89	57.38
Multi 3			
# of blocks	LIRS	DP	UBM
20	8.22	8.63	4.28
35	13.27	13.94	5.21
50	17.79	18.68	7.18
100	27.40	28.77	17.93
200	34.03	35.73	31.78
300	40.31	42.32	37.67
400	44.10	44.98	42.58
500	44.75	45.65	43.97
600	45.52	45.98	44.96
700	46.15	46.61	45.56
800	47.40	48.35	46.52
900	48.37	49.34	47.46
1000	49.36	49.36	48.74

참고문헌

[1] P. Cao, E. W. Felten and K. Li, "Application-Controlled File Caching Policies," Proceedings of the USENIX Summer 1994 Technical Conference, 1994, pp. 171-182.

[2] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky and J. Zelenka, "Informed Prefetching and Caching", Proceedings of the 15th Symposium on Operating System Principles, 1995, pp. 79-95.

[3] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The LRU-K Page Replacement Algorithm for Database Disk Buffering", Proceedings of the 1993 ACM SIGMOD Conference, 1993, pp. 297-306.

[4] T. Johnson and D. Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm", In Proceedings of the 20th International Conference on VLDB, pages 439-450.

[5] D. Lee, J. Choi, J. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "On the Existence of a Spectrum of Policies that Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies", Proceeding of 1999 ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems, May 1999, pp. 134-143.

[6] Song Jiang and Xiaodong Zhang, "LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache Performance", ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, June 15-19, 2002.

[7] G. Glass and P. Cao, "Adaptive Page Replacement Based on Memory Reference Behavior", Proceedings of 1997 ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems, May 1997, pp. 115-126.

[8] Y. Smaragdakis, S. Kaplan, and P. Wilson, "EELRU: Simple and Effective Adaptive Page Replacement", Proceed- ings of 1999 ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems, May 1999, pp. 122-133.

[9] Jongmoo Choi, Sam H. Noh, Sang Lyul Min, and Yookun Cho,"An Implementation Study of a Detection-Based Adaptive Block Replacement Scheme", In Proceedings of the 1999 USENIX Annual Technical Conference, June 1999.

[10] J. M. Kim, J. Choi, J. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "A Low-Overhead, High-Performance Unified Buffer Management Scheme that Exploits Sequential and Looping References", Proceedings of the 4th USENIX Symposium on Operating System Design and Implementation, October 2000, pp. 119-134.

● 저자소개 ●

진재선

2000 성균관대학교 전기전자컴퓨터공학부 학사
2002~현재 성균관대학교 컴퓨터공학부 석사과정
관심분야 : 분산 운영체제, 그리드 컴퓨팅

허의남

1990 부산대학교 공과대학 전산학과 학사
1995 텍사스주립대학교 컴퓨터공학 석사
2002 The Ohio University 컴퓨터공학 박사
2002~현재 서울여자대학교 컴퓨터공학과 교수
관심분야 : 네트워크, QoS, 그리드미들웨어, 실시간시스템

추현승

1988 성균관대학교 이과대학 수학과 학사
1990 텍사스주립대학교 컴퓨터공학 석사
1996 텍사스주립대학교 컴퓨터공학 박사
1998~현재 성균관대학교 컴퓨터공학부 교수
관심분야 : 스토리지시스템, 모바일컴퓨팅, 네트워킹