濟州大學校 産業技術硏究所 論文集, 第10卷, 2號, pp.108-117, 1999. 12
Cheju Nat'l Univ. Res. Insti. Ind.Tech. Jour., Vol.10, No.2 pp.108-117, 1999. 12

LRFU(Least Recently/Frequently Used) 기법의 특성과 적응형 LRFU 기법의 개발

이 동 희*

Characteristics of the LRFU(Least Recently/Frequently Used) policy and design of the adaptive LRFU

Dong-Hee Lee*

ABSTRACT

The performance of computer system can be improved with predicting the future reference behavior. For example, the disk cache or web cache predicts the re-reference probability and maintains disk blocks or web data that are likely to be re-referenced again in near future. The LRU and LFU policies are well known replacement policies that predict re-reference probability of blocks and replace a block with the lowest probability. The LRU policy measures the re-reference probability based on the last reference time, and gives more weight to more recently referenced block than older one. On the other hand, the LFU policy measures the re-reference probability based on the reference frequency, and gives more weight to more frequently referenced block. In this paper, I explain a spectrum of block replacement policies called the LRFU policy that subsumes the LRU and LFU policies. I also propose an adaptive LRFU policy that has a adaptive mechanism to workload evolution. Trace-driven simulations show characteristics of the LRFU policy and performance of the adaptive LRFU policy.

Key words : Block replacement policy, LRU, LFU, LRFU, adaptive LRFU

I. 서 론

컴퓨터 시스템은 많은 부분에서 미래를 예측하여 대처함으로써 성능을 높이고 있다. 대표적인 예로 디스크 캐쉬나 웹 캐쉬가 있는데, 한번 컴퓨터가 참조한 디스크 블록이나 웹 데이터를 메모리의 일부분에 한 디스크 블록이나 웹 데이터를 메모리의 일부분에 마련된 캐쉬에 유지하고, 재 참조되면 캐쉬에서 곧바로 사용하도록 하여 성능을 높일 수 있다[1]. 그러나 이러한 캐쉬는 용량에 제한이 있기 때문에 새로운 디스크 블록이나 데이터를 저장하기 위해서는 기존의 데이터를 캐쉬에서 제거해야 하며, 이 때 되도록이면 미래에 참조될 가능성이 가장 작은 데이터를 제거하는 것이 좋다. 블록 교체 기법은 캐쉬에 있는 블록들 중에서 참조 가능성이 가장 작은 데이터를 선정하는

* 제주대학교 통신·컴퓨터공학부, 산업기술연구소,
Dept. of Telecommunication & Computer Eng., Res. Insti. Ind. Tech., Cheju Nat'l Univ.

기법으로, 블록 교체 기법의 선택은 캐쉬의 성능에 많은 영향을 미친다.

블록 교체 기법으로 널리 사용되는 기법으로 LRU (Least Recently Used)와 LFU(Least Frequently Used) 기법이 있다. LRU 기법은 참조시간을 기준으로 미래의 사용 가능성을 예측하는데, 최근에 참조된 데이터가 오래 전에 참조된 데이터보다 미래에 참조될 가능성이 높다고 생각한다. LFU 기법은 참조된 횟수를 기준으로 미래의 참조 가능성을 예측하는데, 과거에 많이 참조된 데이터가 덜 참조된 데이터보다 미래에 참조될 가능성이 높다고 생각한다. 이처럼 블록 교체 기법은 과거 참조 정보를 이용하여 미래를 예측하며, 주로 블록에 대한 참조 횟수나 참조 시간 등을 이용한다. 참조 횟수나 참조 시간과 같은 정보는 특히 적은 비용으로 비교적 쉽게 획득할 수 있다.

만약 블록 교체 기법이 되도록 많은 과거 정보를 이용하면 좀 더 효율적으로 블록을 교체할 수도 있다. 예를 들면 마지막 K개 참조 시간을 이용하는 LRU-K 기법[2]은 마지막 참조 시간만을 이용하는 LRU(Least Recently Used) 기법에 비하여 우수한 성능을 보인다. 그러나 과거 정보를 되도록 많이 이용하기 위해서는 다음과 같은 연구가 선행되어야 한다. 먼저 많아진 과거 정보를 기반으로 블록의 가치를 판단하기 위한 규칙을 정립해야 한다. 또한 많은 과거 정보를 제한된 기억공간에 유지하고, 이러한 규칙을 효율적으로 구현할 수 있는 교체 알고리즘이 개발되어야 한다.

본 논문에서는 먼저 이러한 LRU 기법과 LFU 기법을 수학적으로 정의하고, 이 두 기법 사이에 스펙트럼이 존재함을 증명하여 LRFU(Least Recently/Frequently Used) 기법[3]을 설명한다. LRFU 기법은 LRU 기법과 LFU 기법을 양 극단으로 하고 이 두 기법 사이에 존재하는 스펙트럼을 의미한다. 아울러 LRFU 기법이 과거의 모든 참조 시간을 고려하면서도 효율적으로 구현이 가능함을 보이고, 실험 결과를 통하여 LRFU 기법의 특성을 설명한다. 그리고 작업 부하에 따라 동적으로 스펙트럼에서 이동하는 적응형 LRFU 기법을 설명하고 모의실험을 통하여 성능을 평가한다. 마지막으로 LRFU 기법의 적용 예를 설명한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 블록 교체 기법을 소개하고 이러한 기법이 이용하는 과거 정보를 비교하며, 3장에서는 LRFU 기법의 기본 구조에 대해 설명한다. 4장에서는 트레이스 기반 모의 실험을 통하여 LRFU 기법의 성능 및 특징을 설명하며, 5장에서 적응형 LRFU 기법을 설명하고 성능을 평가하며, 6장에서 결론을 내린다.

II. 기존 블록 교체 기법들과 이들이 이용하는 과거 정보

널리 이용되는 LRU 기법은 캐쉬에 있는 블록 중 가장 오래 전에 참조된 블록을 교체한다. 즉 LRU 기법은 과거 정보로서 마지막 참조의 최근성을 이용한다. LRU 기법은 작업 집합(working set)의 변화에 잘 적응하여, 많은 경우 비교적 좋은 성능을 보인다. 그러나 LRU-K 기법[2]에서 지적한 바와 같이 LRU 기법은 빈번히 참조되는 블록과 그렇지 못한 블록을 구분하지 못하는데, 그 이유는 마지막 참조 시간이라는 너무 적은 정보만을 이용하기 때문이다.

LFU(Least Frequently Used) 기법은 디스크 블록이 참조된 횟수를 기반으로 블록의 가치를 판단한다. 이 기법은 작업 부하가 변하지 않는 경우 좋은 성능을 보인다. 그러나 참조의 최근성을 고려하지 않기 때문에 과거에 빈번히 참조된 블록들이 캐쉬를 점거하여, 최근에 빈번히 참조되는 블록이 캐쉬에서 충분한 시간동안 유지되지 못하는 캐쉬 오염(cache pollution) 현상이 발생한다.

LRU와 LFU 기법의 문제점을 해결하기 위한 연구가 있어 왔는데, 이들은 모두 되도록 많은 과거 정보를 이용하여 캐쉬 성능을 향상시키고 있다. LRU-K 기법[2]은 마지막 참조 시간이 아니라 마지막에서 K번째 참조 시간을 기준으로 블록을 교체한다. 즉 LRU-K 기법은 마지막 K개의 참조를 바탕으로 블록에 대한 참조 밀도를 유추하고 이를 기준으로 블록의 가치를 판단한다. 논문[2]에 따르면 참조 빈도를 고려하는 LRU-K 기법이 마지막 참조의 최근성만을 고려하는 LRU 기법보다 좋은 성능을 보인다.

순차 참조가 발생할 때 성능이 저하되는 LRU 기법

의 단점을 보완하기 위하여 2Q[4]와 sLRU(Segmented LRU) 기법[5]이 제안되었다. 이들은 모두 임시 큐를 이용하여 순차 참조되는 블록을 다른 블록과 분리시킨다. 블록이 처음으로 참조되면 먼저 임시 큐에 진입하고, 블록이 임시 큐에 있는 동안 재 참조되어야 장시간 유지되는 큐에 진입할 수 있다. 2Q, sLRU와 LRU-K 기법들은 모두 2 개에서 K 개의 참조 시간을 기준으로 블록을 교체하여 LRU나 LFU 기법의 문제를 해결하고 있다.

디스크 캐싱 정책은 많은 분야에서 연구 대상이 되어 왔다[2,4-11]. 특히 데이터베이스분야에서 많은 연구가 있었는데, 대부분 질의 최적화 과정에서 파생되는 정보를 이용하기 때문에 일반적인 파일 캐싱 정책에 응용할 수 없다[11]. 또다른 연구로서 사용자 또는 응용 프로그램 수준에서 캐싱 정책과 관련된 정보를 제공하는 방법으로 상당한 수준의 성능 향상을 제공할 수 있다[7,8].

III. LRFU(Least Recently/Frequently Used) 블록 교체 기법

본 장에서는 LRFU 블록 교체 기법을 소개한다. 먼저 LRFU 기법의 기본 구조를 설명하고, 다음으로 LRFU 기법이 LRU 기법과 LFU 기법을 포함하고 있음을 보인다. 그리고 LRU 기법과 LFU 기법 사이에 스펙트럼이 존재함을 설명한다. 마지막으로 LRFU 기법을 효율적으로 구현할 수 있도록 하는 속성들을 설명한다.

3.1. LRFU 기법의 기본 구조

LRFU 기법에서 블록의 재 참조 가능성은 블록의 가치로 표현되는데, 디스크 블록의 가치는 블록에 대한 참조가 발생할 때마다 증가하며 시간이 지남에 따라 감소한다. 이렇게 시간에 따라 감소하는 비율은 감소 함수 $F(x)$로 계산되는데, x는 참조의 최근성을 의미하며 참조가 발생한 시간과 현재와의 시간 간격이다. 예를 들어 블록 b가 3, 5 그리고 8이라는 시간에 참조되고 현재 시간이 10이라면 블록 b의 가치 ($V_{t_c}(b)$)는 다음과 같다.

$$V_{t_c}(b) = F(10-3) + F(10-5) + F(10-8) = F(7) + F(5) + F(2).$$

과거의 참조보다 최근의 참조에 가중치를 두기 위하여 $F(x)$는 x가 클수록 감소하는 함수이다. 시간이 흘러감에 따라 가치를 감소시키는 감소 함수 $F(x)$는 LRFU 기법의 교체 행태를 결정한다. 예를 들면 시간이 흘러도 참조의 가치가 전혀 감소하지 않는 극단적인 경우로서 모든 x에 대하여 $F(x) = 1$이면 블록의 가치는 바로 블록의 참조 횟수가 되며, 가치가 가장 작은 블록을 교체하는 LRFU 기법은 LFU 기법과 같게 된다. 또 다른 극단적인 경우로 LRFU 기법이 LRU 기법과 동일하게 교체하는 예를 살펴보자. 먼저 블록 A가 시간 0부터 $t-1$까지 매 시간마다 참조되었고, 블록 B는 시간 t에 단 한번 참조되었으며 현재 시간이 $t_c(\geq t)$라고 하자. 비록 블록 A가 블록 B보다 많이 참조되었더라도 블록 B가 A보다 더 최근에 참조되었기 때문에, LRFU 기법이 LRU 기법과 같아지기 위해서는 블록 B의 가치($V_{t_c}(B)$)가 블록 A의 가치($V_{t_c}(A)$)보다 커야 한다. 이를 수식으로 표현하면

$$V_{t_c}(B) = F(t_c - t) > \sum_{t'=0}^{t-1} F(t_c - t') = V_{t_c}(A)$$

가 되며, 이 식을 조금 더 일반화하면 LRFU 기법이 LRU 기법과 동일하게 블록을 교체하는 조건이 된다.

$$\forall i \quad F(i) > \sum_{j=i+1}^{\infty} F(j).$$

LRFU 기법은 LFU와 LRU 조건을 모두 만족하는 함수인 $F(x) = (\frac{1}{2})^{\lambda x}$를 감소 함수로 사용한다. 이 감소 함수에서 λ는 제어 변수로 0 부터 1 사이의 값을 가진다. 먼저 λ가 0인 경우 $F(x) = 1$이 되어 LRFU 기법은 LFU 기법과 동일하게 된다. λ가 1이면 $F(x) = (\frac{1}{2})^{x}$이며 이 경우 LRFU 기법은 LRU 기법과 동일하다. Fig. 1은 λ의 변화에 따라 $F(x) = (\frac{1}{2})^{\lambda x}$ 함수를 보여준다. Fig. 1에서

Spectrum (Recency/Frequency)로 표현된 부분에서 LRFU 기법은 기존의 LRU 및 LFU 기법과 다르게 되며 LRFU 고유의 영역이다.

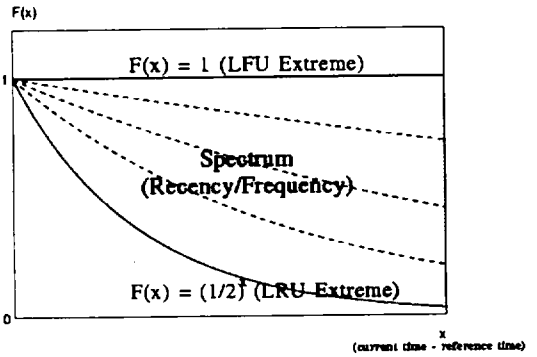


Fig. 1 Spectrum of LRFU according to the function $F(x)=(\frac{1}{2})^{\lambda x}$ where x is (current_time - reference_time).

LRFU 기법은 블록의 가치를 계산하면서 과거의 모든 참조 시간을 요구하며, 모든 블록의 모든 참조 시간을 기록하기 위해서는 무한한 기억 장소가 필요하다. 또한 앞에서 설명한 바와 같이 블록의 가치는 시간이 지남에 따라 변화(감소)한다. 이러한 점들은 이 기법을 비현실적으로 만들 수 있다. 그러나 다음 속성은 이 기법을 현실적으로 가능하게 한다. 감소 함수 $F(x)=(\frac{1}{2})^{\lambda x}$는 $F(x+y) = F(x)F(y)$ 와 같은 성질을 만족하여, 블록에 대한 k번째 참조가 발생한 시점(t_{b_k})에서 블록의 가치($V_{t_{b_k}}(b)$)는 블록에 대한 k-1번째 참조 시간($t_{b_{k-1}}$)과 그 때의 블록 가치 ($V_{t_{b_{k-1}}}(b)$)로부터 계산될 수 있다.

$$V_{t_{b_k}}(b) = F(0) + F(t_{b_k} - t_{b_{k-1}}) V_{t_{b_{k-1}}}(b).$$

마찬가지로 블록이 k 번째 참조된 후 현재 시간(t_c) 까지 참조되지 않았다면, 현재 시간에서 블록의 가치는 다음과 같이 k 번째 참조 시간에서의 블록 가치로부터 계산된다.

$$V_{t_c}(b) = F(t_c - t_{b_k}) * V_{t_{b_k}}(b)$$

이러한 성질로 인하여 블록의 마지막 참조가 발생한 시점과 그 때의 블록 가치만을 유지하면 그 이후 어느 시점에서도 블록의 가치를 구할 수 있다. 따라서 캐쉬의 모든 블록은 자신의 과거 참조 정보를 기록하기 위하여 2개의 값, 즉 마지막 참조 시점과 그 당시의 블록 가치만을 유지하면 된다. 또한 감소 함수 $F(x)$가 $F(x+y) = F(x)F(y)$ 성질을 만족하면 참조되지 않는 블록들 간에 가치의 대소 관계는 시간이 지나도 변하지 않는다. 결국 블록이 참조될 때만 가치에 따른 블록의 정렬이 필요하다.

3.2. LRFU 기법의 시간 복잡도

LRFU 기법은 힙(heap) 자료 구조[13]를 사용하여 블록들을 가치에 따라 정렬하며 이 경우 최대로 O(log n)의 시간 복잡도를 가진다(n은 캐쉬에서 유지되는 블록의 개수)[3]. 그런데 이러한 시간 복잡도는 LRU 기법이 가지는 O(1)의 시간 복잡도와 비교하여 크다. 그러나 다음에 설명할 경계 시간차는 LRFU 기법이 더 효율적으로 구현될 수 있도록 한다.

앞에서 설명한 LRU 조건은 경계 시간차의 존재를 암시하고 있다. 즉 두 블록의 마지막 참조 시간의 차이가 1보다 크거나 같으면, 나중에 참조된 블록의 가치가 항상 크다. 이처럼 다른 조건에 관계없이 가치의 대소 관계를 결정짓는 마지막 참조 시간의 차이가 경계 시간차이다.

이러한 경계 시간차의 존재는 LRFU 기법이 더 효율적으로 구현될 수 있도록 하는 단서를 제공한다. 캐쉬에서 현재 참조되는 블록의 가치보다 큰 가치를 가질 수 있는 블록의 개수는 경계 시간차와 같거나 작다. 왜냐하면 마지막 참조가 경계 시간차 이전에 발생한 블록의 가치는 현재 참조되는 블록의 가치보다 작으며, 그 이후에 참조된 블록만이 현재 참조되는 블록의 가치보다 클 수 있다. 그런데 매 시간에 하나의 블록만 참조된다면, 경계 시간차 이후에 참조된 블록의 최대 개수는 경계 시간차가 된다.

이러한 경계 시간차는 λ값에 따라 달라진다. 앞에서 설명한 바와 같이 LRU 극단에서 경계 시간차는 1이다. 그리고 LFU 극단 쪽으로 이동할수록 경계 시간차는 커지며, 다음과 같은 식으로 계산할 수 있다.

성질 1 감소 함수가 $F(x)=(\frac{1}{2})^{\lambda x}$일 때 다음과 같은 경계 시간차 d_{th}가 존재한다.

$$\forall\ d \geq d_{th},\ F(0) > \sum_{i=d}^{\infty} F(i).$$

이를 만족하는 경계 시간차 d_{th} 값은

$$\left\lceil \frac{\log_{\frac{1}{2}}(1-(\frac{1}{2})^{\lambda})}{\lambda} \right\rceil$$ 이다(증명은 논문[3] 참고).

성질 1에서 $F(0)$는 현재 시간(t_c)에 참조되는 블록이 가질 수 있는 가치의 최소 값이다. 즉 현재 시간에 처음으로 참조된 블록의 가치다. 그리고 $\sum_{i=d}^{\infty} F(i)$는 아주 오래 전부터 (현재 시간 - d) 시간까지 무수히 참조되고 그 이후로 한 번도 참조되지 않은 블록의 가치로서, 마지막 참조가 $t_c - d_{th}$ 이전에 발생한 블록이 가질 수 있는 가치의 최대 값이다. 이 식에 따르면 현재 시간에 참조되는 블록의 가치는 마지막 참조가 $t_c - d_{th}$ 시간 이전에 발생한 블록의 가치보다 크며, 당연히 어떤 블록의 가치가 현재 시간(t_c)에 참조되는 블록의 가치보다 커지려면 마지막 참조가 $t_c - d_{th}$ 시간 이후에 발생했어야 한다. 그리고 매 시간에 단 하나의 블록만이 참조되기 때문에 현재 시간에 참조되는 블록의 가치보다 큰 가치를 가지는 블록의 개수는 d_{th}개보다 작거나 같다.

이 성질은 힙에 유지해야 하는 블록의 개수를 줄여준다. 특히 가장 최근에 참조된 d_{th}개의 블록만 힙으로 유지하고, 나머지 블록은 단순히 연결 리스트로 유지하면 된다. Fig. 2는 최근에 참조된 d_{th}개의 블록을 힙으로 유지하고 나머지 블록을 연결 리스트로 유지하는 디스크 캐쉬 구성을 보여준다. 캐쉬에 존재하지 않는 새로운 블록이 참조되면 Fig. 2(a)와 같이 연결 리스트에서 가장 뒤에 있는 블록이 교체되고, 힙에서 가치가 가장 작은 블록(힙의 루트 블록)이 연결 리스트로 이동하며, 새로 참조되는 블록은 힙에 삽입되고, 블록의 가치에 따라 정렬되어 힙의 속성을 만족하도록 한다. 힙에 존재하는 블록이 참조되면 새로운 가치를 계산하고, 새로운 가치에 따라 힙의 속성을 만족하도록 블록을 재 정렬한다(Fig. 2(b)). 연결 리스트에 존재하는 블록이 참조되면 힙에서 가치가 가장 작은 블록이 연결 리스트로 이동하고, 참조되는 블록이 힙에 삽입되어, 힙의 속성을 만족하도록 블록의 가치에 따라 정렬된다(Fig. 2(c)).

d_{th} 값이 λ에 따라 변한다는 점에 주목할 필요가 있다. λ가 1일 때 d_{th} 값은 1이 되며, 따라서 힙을 사용하지 않고 모든 블록을 연결 리스트로 구성할 수 있다. 이 경우 시간 복잡도는 LRU와 같은 O(1)이다. λ가 감소할수록 d_{th}값은 증가하여 λ가 0일 때 d_{th} 값은 ∞가 된다. 이 경우 모든 블록을 힙으로 구성해야 하며, 시간 복잡도는 LFU 기법과 같은 O(log n)이다(n은 캐쉬에서 유지되는 블록의 개수). 종합하면 LRFU 기법의 시간 복잡도는 LRU 극단에서 O(1)이며 λ가 감소할수록 시간 복잡도가 증가하다가 LFU 극단에서 O(log n)이 된다.

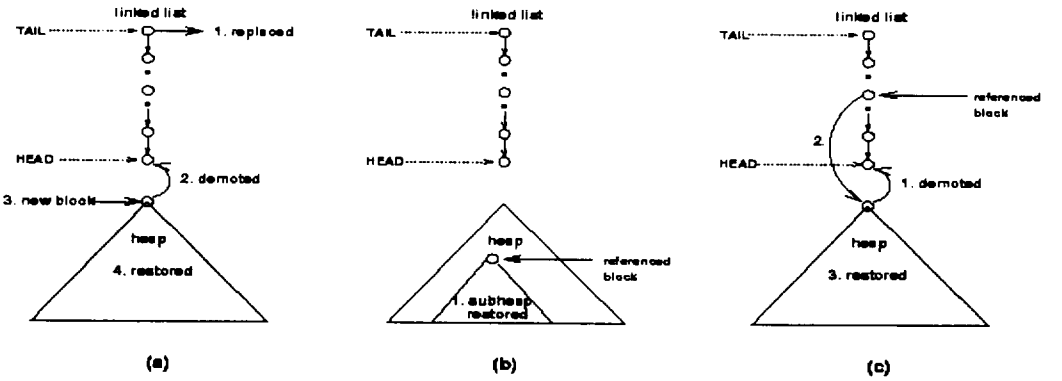


Fig. 2 Optimized implementation of the LRFU policy

3.3. 연관 참조를 고려한 LRFU 기법

연관 참조에 대한 개념은 정확히 정의되지 않았으며 실험[2,6]에 의하여 그 효과가 제기되어 왔다. 많은 경우 하위 레벨의 디스크 블록에 대한 참조보다 트랜잭션과 같은 상위 레벨 사건의 참조 빈도 및 시간 정보가 미래를 예측하는데 더 효과적이며, 연관 참조를 검출하여 디스크 블록에 대한 참조로부터 상위 레벨 사건의 발생을 단순하고 효율적으로 유추할 수 있다.

LRFU 기법은 연관 참조 개념을 자연스럽게 통합할 수 있다. 연관 참조를 고려하는 경우 서로 연관된 참조들을 하나의 비 연관 참조로 간주하고, 비 연관 참조들만을 대상으로 블록의 가치를 계산한다. 연관 참조들을 하나의 비 연관 참조로 변환하기 위하여 다음과 같은 마스크 함수 $G_c(x)$를 도입하였다.

$$G_c(x)=\begin{cases}0 : x \le c\\ 1 : x > c\end{cases}$$

위 식에서 c는 연관 참조기간으로 어떤 블록이 이 기간 안에 다시 참조되면 이전의 참조와 다음 참조는 서로 연관된 참조로 간주하여, 이전 참조는 블록의 가치에 영향을 미치지 않고 마지막 참조만 블록의 가치에 영향을 미친다. 비 연관 참조만을 대상으로 하는 블록의 가치($V'_{t_c}(b)$)는 다음과 같다.

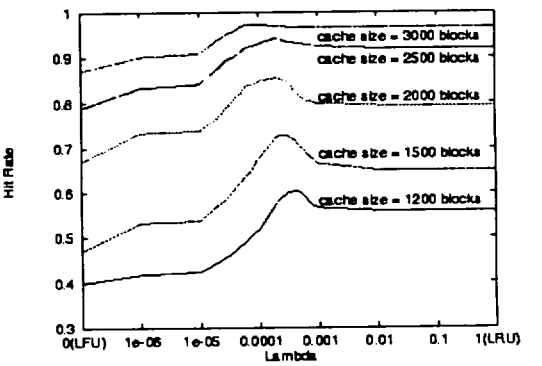
$$V'_{t_c}(b) = F(t_c - t_{b_1}) + \sum_{i=1}^{k-1} F(t_c - t_{b_i}) * G_c(t_{b_{i+1}} - t_{b_i}).$$

연관 참조를 고려할 때 t_{b_k} 시간에 참조되는 블록의 가치($V'_{t_c}(b)$)는 다음과 같이 이전 참조때 구한 블록의 가치로부터 쉽게 구할 수 있으며, LRFU 기법의 효율성에 영향을 미치지 않는다.

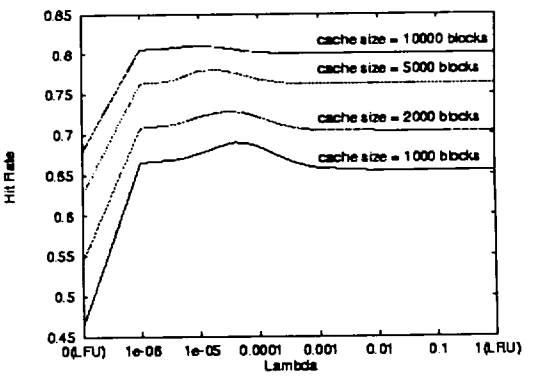
$$\begin{aligned}V'_{t_{b_k}}(b) &= F(t_{b_k} - t_{b_1}) + \sum_{i=1}^{k-1} F(t_{b_k} - t_{b_i}) * G_c(t_{b_{i+1}} - t_{b_i})\\ &= F(0) + F(t_{b_k} - t_{b_{k-1}}) * [F(0) * G_c(t_{b_k} - t_{b_{k-1}}) + V'_{t_{b_{k-1}}}(b) - F(0)]\end{aligned}$$

IV. 실험 결과

본 장에서는 트레이스를 사용하여 LRFU 기법을 모의 실험한 결과를 제시한다. 실험에서는 두 종류의 트레이스를 사용하였다. 첫 번째 트레이스는 스프라이트 네트워크 파일 시스템 트레이스[14]로서 클라이언트 워크스테이션에서 수행중인 응용 프로그램이 참조하는 디스크 블록을 이틀에 걸쳐 기록한 것이다. 실험에서는 가장 많은 블록을 요청하는 54번 클라이언트 워크스테이션을 선정하여, 이 클라이언트의 버퍼 캐쉬 동작을 모의 실험하였다. 54번 클라이언트는 4,822개의 블록에 대하여 141,223번 요청을 한다. 두 번째 트레이스는 DB2 트레이스로서 75,514 블록에



(a) Sprite trace : Client 54



(b) DB2

Fig. 3 Effects of λ on the LRFU policy using Sprite and DB2 traces

대하여 500,000번의 요청이 기록되어 있다. DB2 트레이스는 논문[4]에서 사용되었기 때문에 2Q 기법과 비교를 가능하게 한다.

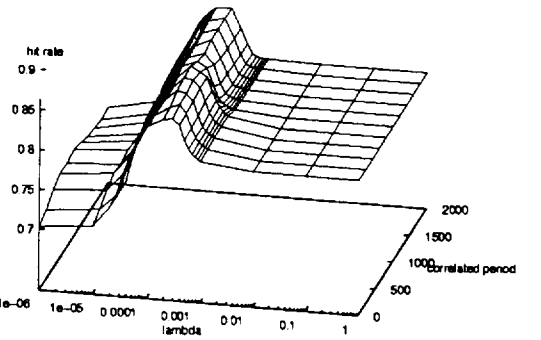
본 논문에서 제안하는 LRFU 기법과 기존에 제안된 LRU-2와 2Q 기법을 비교하기 위하여 LRU-2 기법과 2Q 기법을 구현하였다. 또 비 연관 참조만을 고려할 때 LFU 극단에서 LRFU 기법은 FBR 기법[6]과 같기 때문에 LRFU 기법은 FBR 기법을 포함한다.

4.1. λ의 변화에 따른 LRFU 기법의 캐쉬 적중율

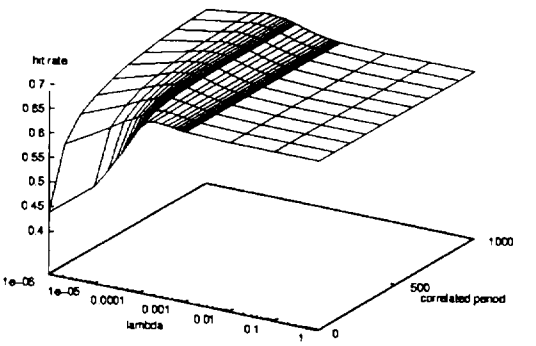
Fig. 3은 λ를 변화시킬 때 LRFU 기법의 캐쉬 적중율을 보여준다. 전체적으로 λ가 0(LFU 극단)일 때 캐쉬 적중율이 가장 작다. λ가 증가하면서 캐쉬 적중율도 증가하여 최고점에 오른 후 다시 감소한다. 그리고 λ가 1(LRU 극단)에 도달할 때까지 캐쉬 적중율은 매우 천천히 감소한다. 그림에서 캐쉬 크기가 증가할수록 가장 좋은 캐쉬 적중율을 보이는 λ 값은 작아진다. 즉 캐쉬 크기가 증가할수록 과거의 참조에 가중치를 더 많이 두어 블록의 가치를 계산해야 하는데, 이는 블록의 단기적인 참조 형태보다 장기적인 참조 형태를 기준으로 블록의 참조 가능성을 결정해야 한다는 것을 의미한다.

4.2. 연관 참조기간 c의 변화에 따른 LRFU 기법의 캐쉬 적중율

Fig. 4는 λ와 연관 참조기간 c를 변화시킬 때 LRFU 기법의 캐쉬 적중율을 보여준다. DB2 트레이스를 사용한 Fig. 4(b)에서 연관 참조기간이 작은 경우 λ값의 변화가 캐쉬 적중율에 미치는 영향이 크다. 특히 LFU 극단에 가까워지면 캐쉬 적중율은 급격히 감소한다. 그러나 연관 참조기간이 커질수록 λ 값에 따라 캐쉬 적중율의 변화가 줄어들며, 완만한 구릉 모양을 보여준다. 또 다른 특징으로 LRU 극단(λ가 1)에서는 연관 참조기간이 캐쉬 적중율에 영향을 미치지 못한다. 하지만 LFU 극단(λ가 0)에서는 연관 참조기간이 캐쉬 적중율에 영향을 미친다. λ가 작을 때 연관 참조기간이 증가하면 캐쉬 적중율도 어느 정도 증가하며, 이 점은 비 연관 참조만을 대상으로 LFU 기법을 적용한 FBR 기법이 LRU 보다 성능이 좋아질 수 있음을 확인한다. 마지막으로 연관 참조기간이 클수록 가장 좋은 캐쉬 적중율을 보이는 λ값은 작아진다. 연관 참조기간은 디스크 참조와 같은 하위 레벨 동작으로부터 상위 레벨 동작의 참조 횟수를 유추하는 역할을 한다. 실험 결과에 따르면 연관 참조기간으로 상위 레벨 동작을 유추하는 경우 참조 횟수에 비중을 두면(LFU 극단 쪽으로 다가갈수록) 더 이득이라는 점을 보여준다. Fig. 4(a)의 경우 λ의 변화는 캐쉬 적중율에 영향을 미치나 연관 참조기간의 변화는 영향을 미치지 않는다. 그 이유는 스프라이트 트레이스의 경우 클라이언트 캐쉬에서 연관 참조가 모두 만족되며, 트레이스에는 이러한 연관 참조들을 대표해서 단 한 번의 비 연관 참조만이 기록되어 있기 때문이다.



(a) Sprite trace : Client 54

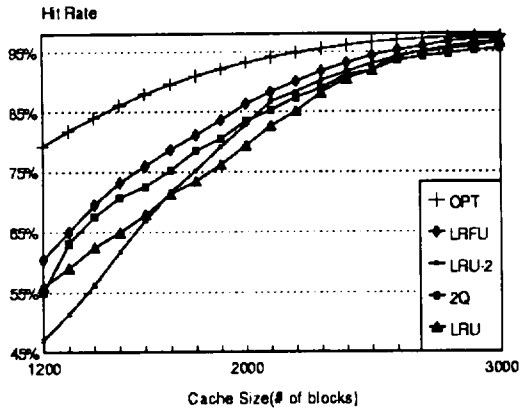


(b) DB2

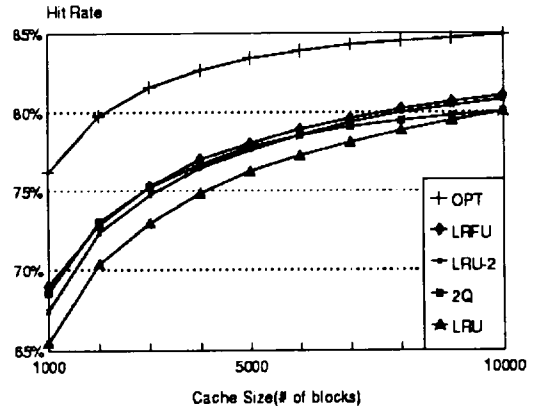
Fig. 4 Combined effects of λ and c on the LRFU policy

4.3. LRFU 기법과 다른 기법과의 성능 비교

Fig. 5는 LRFU, LRU, 2Q, 그리고 LRU-2 기법의 성능을 비교한 것이다. 2Q와 LRU-2 기법의 경우 논문[4]에서 제시한 바와 같이 제어 인자(control parameter)인 연관 참조기간(임시 큐의 크기)을 캐쉬 크기의 20%와 30%로 할 때 성능을 측정하여 좋은 결과를 선택하였다. 마찬가지로 LRFU 기법에서도 제어 인자를 조정하여 성능이 좋은 결과를 선택하였다. 실험 결과에 따르면 대부분 LRFU 기법이 가장 좋고 LRU 기법이 가장 나쁜 성능을 보인다. 그리고 2Q와 LRU-2기법이 LRFU와 LRU 기법 사이에 위치하고 있다. 그러나 스프라이트 트레이스에서 2Q와 LRU-2 기법은 캐쉬 크기가 작을 때 LRU 기법보다 캐쉬 적중율이 떨어지며, 캐쉬 크기가 커지면 캐쉬 적중율이 LRU 기법으로 수렴한다. DB2 트레이스에서도 2Q와 LRU-2 기법은 캐쉬 크기가 커지면 LRU 기법으로 수렴하며, 2Q 기법의 경우 LRU 기법보다 성능이 나쁘다. LRFU 기법은 항상 LRU 기법보다 성능이 좋으며 대부분의 경우 가장 좋은 성능을 보이나, DB2 트레이스를 사용할 때 캐쉬 크기가 작은 몇몇의 경우 2Q나 LRU-2 기법보다 캐쉬 적중율이 약간 떨어진다. 그 이유는 2Q나 LRU-2 기법에서는 블록이 교체되어 캐쉬에서 제거되는 경우에도 일정 기간 제거된 블록의 기록을 유지하여, 블록이 재 참조되면 이 기록을 이용하여 블록을 가치를 판단한다. 그러나 LRFU 기법의 경우 제거된 블록의 기록을 유지하지 않았다. LRFU 기법에서도 이러한 기록을 유지하여 실험한 결과에 따르면 모든 캐쉬 크기에서 LRFU 기법의 캐쉬 적중율이 약간 증가하며, 캐쉬 크기가 작은 경우에도 2Q나 LRU-2기법보다 우수한 성능을 보인다(이 결과는 본 논문에서는 제시하지 않았다).



(a) Sprite trace : Client 54



(b) DB2

Fig. 5 Comparison of LRFU with other policies using Sprite and DB2 traces

V. 적응형 LRFU 기법

λ값이 LRFU 기법의 성능에 영향을 미치기 때문에 적절한 λ값을 결정하는 기법이 요구된다. 본 장에서는 작업부하에 따라 λ값을 결정하는 적응형 LRFU 기법에 대해 설명한다. 적응형 LRFU 기법에서는 매 주기(period)마다 LRFU 기법의 성능 향상 정도를 측정한다. 그리고 이 결과에 따라 λ값을 증가시키거나 감소시킨다. i-1번째 주기에서 λ값이 단위량 만큼 증가했다고 하자. i번째 주기에서는 이 λ값을 사용하며, i번째 주기에서의 성능이 i-1번째 주기의 성능보다 좋았다면, i+1번째 주기를 위한 λ값은 단위량만큼 증가된다. 그러나 i번째 주기에서 LRFU 기법의 성능이 i-1번째 주기보다 나빠졌다면 λ값을 감소시키기 시작한다. 반대로 i-1번째 주기에서 λ값이 단위량 만큼 감소했으며, i번째 주기에서 LRFU 기법의 성능이 i-1번째 주기보다 좋다면, 계속해서 λ값은 감소한다. 그러나 성능이 나빴다면 λ값

은 증가하기 시작한다.

이 기법의 문제점은 LRFU 기법의 성능 향상 정도를 측정하는 방법이 모호하다는 것이다. 단순히 캐쉬 적중률만을 가지고 측정할 수도 있으나, 캐쉬 적중률은 작업부하에 따라 현격히 변화한다. 따라서 캐쉬 적중률이 좋아졌다고 해서, 반드시 새로운 λ값의 영향이라고 할 수는 없다.

이와같이 작업 부하에 따라 현격히 달라지는 캐쉬 적중률 대신에 적응형 LRFU 기법은 LRU 기법을 기준으로 LRFU 기법의 상대적인 성능 향상 정도를 바탕으로 λ값을 변화시킨다. 주기 i에서 적응형 LRFU 기법은 LRFU 기법의 성능 향상률 $(\frac{hit_i^{LRFU} - hit_{i-1}^{LRFU}}{hit_{i-1}^{LRFU}})$과 LRU 기법의 성능 향상률 $(\frac{hit_i^{LRU} - hit_{i-1}^{LRU}}{hit_{i-1}^{LRU}})$을 비교하여, LRFU 기법의 성능 향상률이 LRU 기법의 것보다 크면 i+1번째 주기에서 사용할 λ값을 단위량만큼 계속적으로 증가(이전 주기에서 감소시켰으면 감소)시킨다. 만약 LRFU 기법의 성능 향상률이 LRU 기법보다 작으면 λ값의 증감 방향은 반대가 된다. 즉 이전 주기에서 감소 시켰으면 증가시키며, 증가 시켰으면 감소시킨다. LRU 기법을 기본으로 한 상대적인 성능 향상률을 이용하여 λ값을 결정하는 방법의 장점은 현저하게 달라지는 작업 부하의 특성을 배제할 수 있어, 더 정확하게 λ값의 증감 방향을 결정할 수 있다는 것이다.

λ값을 증가시키거나 감소시키는 단위량은 $\frac{1}{10^{\lfloor \log_{10}\lambda_i \rfloor + 1}}$ 로 계산된다. 예를 들어 λ값이 0.03이면 단위량은 0.01이 되며, λ값이 0.00005이면 단위량은 0.00001이다.

Table 1은 LRU, LRFU, 그리고 적응형 LRFU 기법의 성능을 보여준다. Table 1에서 보면 적응형 LRFU 기법의 성능은 LRU 기법보다 좋으며, LRFU 기법보다는 떨어진다. 그 이유는 다음과 같이 3가지로 설명된다. 첫째, 적응형 LRFU 기법에서 사용하는 λ값은 작업 부하에 따라 계속적으로 변화하며, 최적의 λ값과 다르다. 따라서 최적의 λ값을 사용하는 LRFU 기법보다 성능이 떨어진다. 둘째, 적응형 LRFU 기법에서는 급격히 변화하는 작업 부하의 특성을 배제하기 위하여 LRU 기법과의 상대적인 성능 향상률을 이용하여 λ값을 변화시키는데, 이러한 LRU 기법도 작업 부하의 특성에 어느 정도 영향을 받는다. 셋째, 실험에 사용한 트레이스의 길이가 적응형 LRFU 기법을 실험하기에는 너무 짧았기 때문에, 충분한 적응 기간을 제공하지 못했다. 현재 적응형 LRFU 기법은 작업 부하에 따라 λ값을 동적으로 변화시키며 동작하며, LRU 기법보다 좋은 성능을 보이지만, 더욱 성능을 개선시킬 수 있는 가능성을 많이 가지고 있다. 따라서 이 분야로 많은 연구가 수행되어져야 한다.

Table 1 Results of the adaptive LRFU policy

캐쉬크기	LRU	LRFU (비적응형)	적응형 LRFU
1200	0.5588	0.6049	0.5872
1400	0.6247	0.6952	0.6688
1600	0.6789	0.7601	0.7478
1800	0.7346	0.8112	0.8017
2000	0.7939	0.8634	0.8461
2200	0.8511	0.9009	0.8851
2400	0.9057	0.9317	0.9199
2600	0.9437	0.9530	0.9492
2800	0.9552	0.9672	0.9641
3000	0.9657	0.9726	0.9707

캐쉬크기	LRU	LRFU (비적응형)	적응형 LRFU
1000	0.6544	0.6899	0.6772
2000	0.7038	0.7285	0.7213
3000	0.7295	0.7527	0.7463
4000	0.7483	0.7701	0.7575
5000	0.7625	0.7802	0.7652
6000	0.7725	0.7891	0.7754
7000	0.7809	0.7962	0.7815
8000	0.7885	0.8024	0.7951
9000	0.7949	0.8068	0.7997
10000	0.8006	0.8107	0.8023

VI. 결 론

본 논문에서는 LRU와 LFU 기법을 양 극단으로 하여 그 사이에 존재하는 스펙트럼으로 구성된 LRFU 기법에 대해 기술하였다. LRFU 기법은 제어 변수인

λ를 조절하여 스펙트럼에서 이동하게 되는데, 모의 실험 결과에 따르면 스펙트럼상에 기존 기법들보다 우수한 성능을 보이는 지점이 존재한다. 아울러 작업 부하에 따라 λ값을 동적으로 변화시키는 적응형 LRFU 기법을 제안하고 그 성능을 평가하였다. 적응형 LRFU 기법은 LRU 기법보다는 좋은 성능을 보이나, 최적의 λ값을 사용하는 LRFU 기법보다는 좋지 않은 성능을 보이고 있다. 앞으로 작업부하에 따라 λ값을 결정하는 적응 기법은 많은 연구를 필요로 하고 있다. 현재 LRFU 기법은 웹 캐쉬에도 적용되어 좋은 결과[15]를 보이고 있다. 앞으로도 LRFU 기법을 다양한 분야에 적용하여 좋은 결과를 기대할 수 있을 것이다.

참고문헌

1) M. J. Bach, 1986, *The Design of the UNIX Operating System*, Prentice-Hall, Englewood Cliffs, NJ.

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

3) D. Lee, J. Choi, J. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, 1999, On the Existence of a Spectrum of Policies that Subsumes the LRU and LFU Policies, Proceedings of 1999 ACM SIGMETRICS Conference, pp.134-143.

4) T. Johnson and D. Shasha, 1994, 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm, *Proceedings of the 20th International Conference on Very Large Data Bases*, pp.439-450.

5) R. Karedla, J. S. Love, and B. G. Wherry, 1994, Caching Strategies to Improve Disk System Performance, *IEEE Computer*, Vol.27, No.3, pp.38-46.

6) J. T. Robinson and N. V. Devarakonda, 1990, Data Cache Management Using Frequency-Based Replacement, *Proceedings of the 1990 ACM SIGMETRICS Conference*, pp.134-142.

7) P. Cao, E. W. Felten, and K. Li, 1994, Application-Controlled File Caching Policies, *Proceedings of the Summer 1994 USENIX Conference*, pp.171-182.

8) R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, 1995, Informed Prefetching and Caching, *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pp.141-157.

9) V. Phalke and B. Gopinath, 1995, An Inter-Reference Gap Model for Temporal Locality in Program Behavior, *Proceedings of the 1995 ACM SIGMETRICS/PERFORMANCE Conference*, pp.291-300.

10) W. Effelsberg and T. Haerder, 1984, Principles of Database Buffer Management, *ACM Transactions on Database Systems*, Vol.9, No.4, pp.560-595.

11) C. Faloutsos, R. Ng, and T. Sellis, 1995, Flexible and Adaptable Buffer Management Techniques for Database Management Systems, *IEEE Transactions on Computers*, Vol.44, No.4, pp.546-560.

12) E. G. Coffman, Jr. and P. J. Denning, 1973, *Operating Systems Theory*, Prentice-Hall, Englewood Cliffs, NJ.

13) J. D. Smith, 1989, *Design and Analysis of Algorithms*, PWS-KENT Publishing Company, Boston, MA.

14) M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout, 1991, Measurements of a Distributed File System, *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pp.198-212.

15) H. Bahn, S. H. Noh, S. L. Min, and K. Koh, 1999, Using Full Reference History for Efficient Document Replacement in Web Caches, *Proceedings of the 2nd USENIX Symposium on Internet Technologies & Systems*.