M-ARC: ARC 기반 고성능 멀티레벨 버퍼캐시 알고리즘

박세진 ○ 박찬익
포항공과대학교
{baksejin○, cipark}@postech.ac.kr

M-ARC : ARC based high performance multi-level buffer cache algorithm

Sejin Park○ Chanik Park
POSTECH

요 약

멀티레벨 스토리지 접근은 클라우드 시스템, 가상화 환경, 네트워크 기반 스토리지 등 많은 컴퓨팅 환경에서 널리 사용되고 있다. 이러한 멀티레벨 스토리지의 접근성능을 향상시키려면, 되도록 하위 레벨의 스토리지로 요청이 일어나지 않게 하는 것이 중요하며, 이는 각 레벨의 버퍼캐시 성능이 큰 영향을 미친다. 다양한 버퍼캐시 알고리즘들 중 ARC 알고리즘은 동작의 간결성과 고성능으로 인해, 많은 워크로드에서 가장 좋은 성능을 보이는 캐시 알고리즘으로 알려져 있다. 그러나, ARC 알고리즘은 2차 레벨 버퍼캐시에서는 좋은 성능을 보이지 않는데, 이는 ARC 알고리즘이 멀티레벨 캐시의 특성을 반영하지 못하고 있기 때문이다. 본 논문에서는 멀티레벨 캐시의 특성과 이를 반영한 M-ARC 라는 멀티레벨 버퍼캐시 알고리즘을 제안한다. 제안하는 알고리즘은 기존 ARC 에 비해 약 2배 이상 향상된 성능을 보여주고 있다.

1. 서 론

버퍼캐시에는 일반적으로 자주 사용되는 데이터 블록들이 위치하고 있어, 물리 디스크에 접근하지 않고, 메모리 접근만으로 데이터 블록을 사용할 수 있게 하여, 시스템의 I/O 성능에 큰 영향을 미친다. 개별 미디어 의 성능차이가 있지만, DRAM 성능과 디스크 접근시간의 차이는 일반적으로 10,000 배 ~ 100,000 배 정도로 성능 차이가 크게 나기 때문에 일반적인 스토리지 시스템에서 버퍼캐시 성능은 중요하다. 이 버퍼캐시의 성능을 높이기 위하여 다양한 알고리즘들이 제안되었다.[1,2,3,4,5,6] 클라우드 기술, 가상화 기술, 네트워크 스토리지 기술 등 계층적으로 스토리지를 사용하는 멀티레벨 스토리지 환경이 점차 늘어가고 있는데 이런 환경에서는 멀티레벨 캐시의 워크로드 특성을 반영한 버퍼캐시가 필요하다.

2. 기초 지식 및 관련 연구

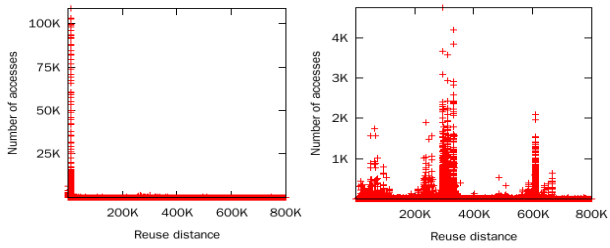
멀티레벨 캐시의 워크로드는 최상위 응용 프로그램에서 요청한 것이 아니라, 멀티레벨 캐시 계층상 바로 상위 레벨의 캐시에서 요청한 것이다. 예를 들어 가상화 환경의 경우(Xen Hypervisor, HVM 의 경우), 게스트 운영체제에서 구동되고 잇는 응용 프로그램이 디스크의 데이터블록을 요청할 경우, 다음과 같은 과정을 거쳐서 최종적으로 응용프로그램이 블록을 접근하게 된다.

0. 응용 프로그램의 블록 요청
1. 게스트 운영체제 버퍼캐시 탐색
2. 호스트 운영체제 버퍼캐시 탐색
3. 물리 디스크 접근

응용 프로그램이 요청한 블록은 우선 1 게스트 운영체제의 버퍼캐시에 있는지 확인하게 되고, 없을 경우, 2. 호스트 운영체제의 버퍼캐시를 확인한다. 여기에도 없을 경우, 3. 물리 디스크에 접근해서 해당 블록을 계층적으로 최 상위 응용 프로그램까지 전달하게 된다.

멀티레벨 스토리지의 관점에서는 게스트 운영체제 버퍼캐시는 1차 레벨 버퍼캐시, 호스트 운영체제 버퍼캐시는 2차 레벨 버퍼캐시가 된다. 따라서, 2차 레벨 버퍼캐시의 워크로드는 응용 프로그램의 블록 요청이 아니라, 응용 프로그램의 요청 중 1차 레벨 버퍼캐시에서 Miss 가 난 요청들이 들어오게 된다. 이 경우, 2차 레벨에서의 접근 패턴은 1차 레벨캐시와 달라지게 된다. 이는 블록의 재사용 거리 그래프를 통해 설명되는데, 블록의 재사용 거리란, 한 블록이 접근된 시점부터 다시 접근 되는 시점까지 사이에 있는 요청된 블록의 개수이다. 즉 블록의 재 사용 거리가 짧을수록 사용 빈도가 높은 블록이 된다. 그림 1은 MSR-Cambridge usr0[7] 워크로드를 구동했을 경우 1차 캐시와, 2차 캐시의 블록 재사용 거리를 나타낸 그래프이다. 2차 레벨 캐시는 1차 레벨에서 동일한

워크로드를 64 MB 크기를 가진 LRU 캐시에서 구동했다. 1차 캐시의 블록 재사용 거리 그래프는 거의 0에 가까운 재사용 거리에서 대 다수의 접근이 이루어졌음을 알 수 있고, 2차 캐시의 블록 재사용 거리 그래프는 약 300K 주변에서 가장 많은 접근이



(a) 1 차캐시 (b) 2 차캐시

그림 1. 1 차 캐시와 2 차 캐시의 블록 재사용 거리 그래프

이루어졌음을 알 수 있다.

즉 2차 레벨버퍼캐시의 워크로드는 재사용 거리가 길어지는데, 재사용 거리가 짧은 워크로드들의 경우, 상위 레벨 캐시에서 이미 캐시에 저장되어있기 때문이다. 멀티레벨 버퍼캐시에 대한 연구는 현재 많지 않으며, 대표적인 것으로 Multi-Queue (MQ) 알고리즘[2.3] 이 있다. 이 알고리즘 역시 2차 레벨의 워크로드 특성을 반영하기 위해 여러 개의 LRU 큐를 사용 빈도에 따라 순위를 정해 유지시킨다. 이 알고리즘에서는 시간적 지역성을 확보하기 위해서, 각 캐시 엔트리들이 만료시간이라는 값을 유지하는데, 이를 확인하기 위해서는 모든 LRU 큐를 다 순회해야 하는 단점을 가진다. LIRS[6] 알고리즘은 각 엔트리의 재사용 거리를 기반으로 동작하고 있지만, 멀티레벨의 특성을 반영하지 않고, 재사용 거리가 짧은 블록 순으로 캐시를 유지 시킨다. 연구[5] 에서는 각 레벨에서 효과적으로 캐시를 유지시키기 위해서, DEMOTE라는 새로운 SCSI 명령어를 제안하기도 했으며, 파일 시스템의 의미적 분석을 통해 각 레벨의 캐시를 효과적으로 유지시키는 XRAY[4] 와 같은 연구도 진행되었다. 이 기법은 캐시는 효과적으로 유지되지만, 각 파일 시스템의 inode 정보를 매번 확인해야 하는 큰 성능적인 오버헤드를 내포하고 있다.

3. 설계

3.1 ARC[1] 알고리즘

ARC 알고리즘은 다양한 워크로드에서 가장 좋은 성능을 나타내는 것으로 알려져 있다.[1] ARC는 두 개의 LRU Stack 을 이용하여 구현이 된다. 전체 캐시 크기가 C 일 때, 초기 값으로 두 개의 LRU 스택 LRU0, LRU1은 각각 C/2 길이를 가지며, 각각 히스토리 버퍼 HB0, HB1을 가진다. 히스토리 버퍼는 실제 블록은 저장하지 않고 블록번호만 저장하여 접근한 기록을 유지한다. LRU0과 LRU1은 워크로드에 따라 크기가 동적으로 변경되며 다음의 룰을 따른다.

1. 최초 엔트리는 LRU0 로 삽입
2. 캐시 엔트리가 밀려나가서 LRU0를 크기를 넘어서는 순간 HB0으로 들어가며 데이터 블록은 제거되며 블록번호만 유지.
3. LRU1 + LRU0 = C
4. LRU0 + HB0 = C , LRU1 + HB1 = C
5. 만약 HB0 에 있는 엔트리에서 접근이 일어나면 해당 엔트리는 LRU1으로 이동하며, 이때, LRU0 크기는 +1, LRU1 크기는 -1
6. 만약 HB1 에 있는 엔트리에서 접근이 일어나면 해당 엔트리는 LRU1으로 이동하며, LRU1 크기는 +1, LRU0 의 크기는 -1.
7. LRU1 에서 밀려나간 엔트리는 HB1으로 삽입되며, 데이터 블록은 제거되고, 블록번호만 유지.

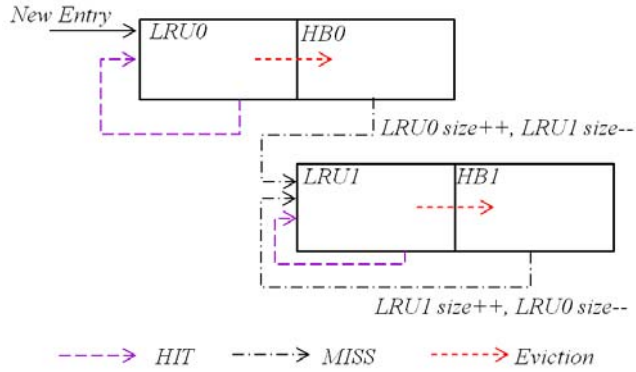


그림 2. ARC 알고리즘

3.2 2차 레벨에서 ARC 알고리즘

이 ARC 알고리즘은 최대 2C 크기까지의 접근 기록이 히스토리 버퍼를 이용해 기록이 된다. 하지만, 2절에서 설명한 것처럼 2차 레벨 버퍼캐시의 재 접근 거리가 길기 때문에, 대 부분의 캐시 엔트리가 재 접근 되기 전에 캐시에서 빠져나가는 문제를 내포하고 있다.

3.3 M-ARC 알고리즘

그림 3은 M-ARC 알고리즘을 나타내고 있다. 이것은, ARC 알고리즘은 확장한 것으로 멀티레벨과 같은 환경에서 재 접근 거리가 긴 워크로드가 캐시에 들어올 경우, 이를 효과적으로 처리해주는 구조를 가진다. 재 접근 거리가 긴 워크로드를 처리하기 위해 HB를 확장시키고 (HB1~ HBn), HBn 에서 접근이 될 경우, 별도의 LRUn 스택으로 해당 엔트리를 위치 시켜서, 재사용 거리가 길어질 경우에도 캐시엔트리가 캐시 내에 유지될 수 있도록 해준다. n 값은 변경할 수 있는 값으로 본 연구에서는 n=10 에서 가장 좋은 Hitratio 값을 확인 하였다. 히스토리 버퍼가 많아지면 메모리 사용량도 동시에 늘어나지만, 이 버퍼에는 데이터가 들어가지 않고, 메타데이터(블록번호)만 유지하기 때문에 공간적 오버헤드는 크지 않다. (n=10, 4KB 블록 크기, 4KB 메타데이터 크기를 가질 때의 공간적 오버헤드는, 전체 메모리 공간의 0.28% 사용)

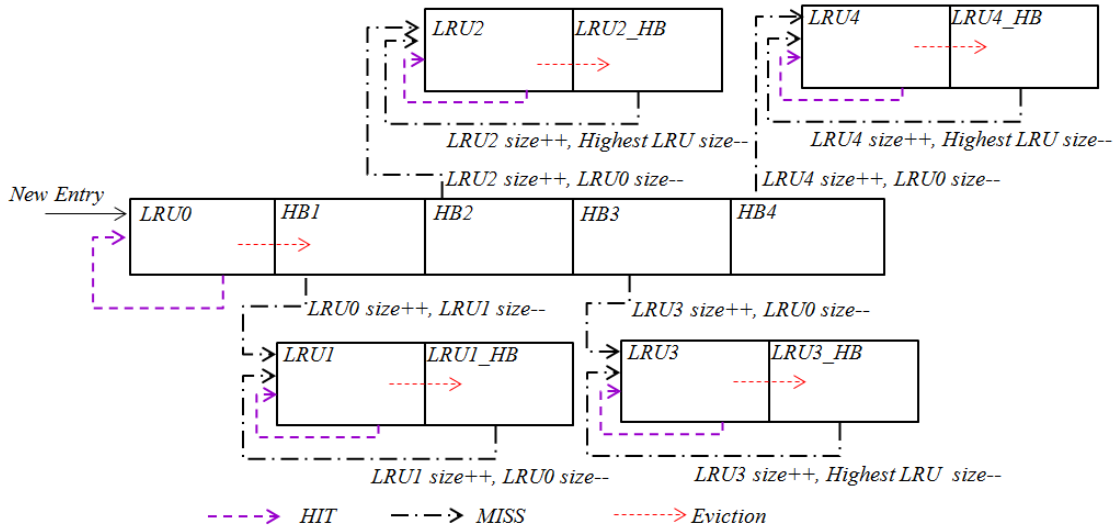


그림3. M-ARC 구조

M-ARC는 다음의 캐시 크기 변경 규칙을 따른다.

1. HB1 에서 접근: LRU0++, LRU1--
2. LRU1_HB 에서 접근: LRU1++, LRU0 --
3. HBn, n>1 에서 접근: LRUn++, LRU0 --
4. LRUn_HB, n>1 에서 접근: LRUn++, 현재 가장 크기가 큰 LRU--

본 연구에서 제안하는 알고리즘은 LRU 스택에 최소의 추가 연산만을 통해 쉽게 구현이 가능하여, O(1)의 시간 복잡도를 가지는 캐시 교체를 할 수 있다.

4. 평가

평가는 시뮬레이터를 기반으로 하였으며, 사용한 워크로드는 MSR Usr0, proj0[7] 이며. 1차캐시는 Usr0의 경우 64MB, proj0의 경우 128MB의 크기를 가지는 LRU 캐시를 사용하였으며, 1차 캐시에서 내려오는 블록 요청들을 LRU, ARC, M-ARC 알고리즘을 구동하는 다양한 크기의 (16MB~64MB) 2차 캐시로 삽입하여 나타나는 2차 캐시의 Hitratio를 비교하였다.

표1. Usr0 (64MB) 워크로드의 2차 캐시 Hitratio 비교

	16MB	32MB	64MB
LRU	0	0	0.005
ARC	0	0	0.023
M-ARC	0.003	0.017	0.039

표2. proj0 (128MB) 워크로드의 2차 캐시 Hitratio 비교

	16MB	32MB	64MB
LRU	0	0	0
ARC	0	0	0
M-ARC	0.003	0.017	0.039

5. 결 론

본 연구에서는 멀티레벨 버퍼캐시에 요청되는 워크로드의 재 사용 거리가 멀다는 특성을 이용해 ARC 알고리즘을 기반으로 하는 새로운 캐시 기법을 제안하였다. 제안하는 알고리즘은 기존 ARC 알고리즘보다 히스토리 버퍼 공간을 더 사용한다는 단점은 있으나, 2차 캐시에서 현재 사용되고 있는 ARC, LRU보다 훨씬 높은 Hitratio를 보여주고 있다. 또한, LRU스택 기반으로 O(1)의 시간 복잡도를 가진 간단한 구현이 가능하다는 장점을 가진다.

6. Acknowledgement

이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원(No. 2012-0005286) 및 지식경제부 및 정보통신 산업 진흥원의 "IT명품 인재 양성 사업"의 (C1515-1121-0003) 연구결과로 수행 되었음

<참고문헌>

[1] Megiddo, N. and Modha, D.S.,Outperforming LRU with an adaptive replacement cache algorithm, IEEE Computer, Vol 37, No. 4, p58-65, 2004
[2] Y. Zhou, Z. Chen, and K. Li, "Second-level Buffer CacheManagement," IEEE Transactions on Parallel and Distributed Systems, Vol. 15, No. 7, July, 2004.
[3] Y. Zhou, J.F. Philbin, and K. Li, "Second-level Buffer Cache Management," In Procedings of the 2001 USENIX Annual Technical Conference, 2001.
[4] L.N. Bairavasundaram, M. Sivathanu ,A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, "X-RAY: A Non-Invasive Exclusive Caching Mechanism for RAIDs,"Proceedings of the 31st annual international symposium on Computer architecture, p.176, June 19-23, 2004.
[5] T.M. Wong and J. Wilkes, "My Cache or Yours? Making Storage More Exclusive," In Proceedings of the USENIX Annual Technical Conference, 2002.
[6] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," in Proc. ACM SIGMETRICS Conf., 2002.
[7]MSR Cambridge workloads www.snia.org