HDD와 SSD의 혼합형 저장 시스템을 위한 절전형 버퍼 캐쉬 관리

(Low-power Buffer Cache Management for Mixed HDD and SSD Storage Systems)

강 효 정 † 박 준 석 ††
(Hyojung Kang) (Junseok Park)
고 건 ††† 반 효 경 ††††
(Kern Koh) (Hyokyung Bahn)

요 약 본 논문은 하드디스크와 NAND 플래시메모리를 동시에 사용하는 저장 시스템 환경에서 전력 소모를 최소화하는 버퍼 캐쉬 관리 기법을 제안한다. 저장장치별 전력 소모율과 입출력 연산 종류(읽기 또는 쓰기) 및 블록의 재참조 가능성(최근성 및 빈도)을 통합적으로 고려하는 버퍼 캐쉬 관리 기법의 설계로 저장 시스템의 전력 소모량을 평균 18.0%, 최대 58.9%까지 줄일 수 있음을 보인다.

키워드 : 저전력, 버퍼 캐쉬, 교체 알고리즘, 플래시메모리, 하드 디스크

Abstract *A new buffer cache management scheme that aims at reducing power consumption in mixed HDD and NAND flash memory storage systems is presented. The proposed scheme reduces power consumption by considering different energy-consumption rate of storage devices, I/O operation type (read or write), and reference potential of cached blocks in terms of both recency and frequency. Simulation shows that the proposed scheme reduces power consumption by 18.0% on average and up to 58.9%.*

Key words : Low-power, Buffer Cache, Replacement Algorithm, Flash Memory, Hard Disk

· 본 논문은 2008-2009 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(과제책임자 고건: KRF-2008-314-D00344)
(과제책임자 반효경: No.2009-0077659)
· 이 논문은 제36회 추계학술발표회에서 'HDD와 SSD로 구성된 저장 시스템을 위한 절전형 버퍼 캐시 관리'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 이화여자대학교 컴퓨터공학과
7daysinjune@naver.com
†† 학생회원 : 서울대학교 컴퓨터공학부
redo@oslab.snu.ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
kernkoh@oslab.snu.ac.kr
†††† 종신회원 : 이화여자대학교 컴퓨터공학과 교수
bahn@ewha.ac.kr
(Corresponding author임)

논문접수 : 2009년 12월 24일
심사완료 : 2010년 2월 10일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제4호(2010.4)

1. 서 론

NAND 플래시메모리는 가볍고 물리적 충격에 강하며 전력 소모가 적고 접근 속도가 빨라 임베디드 시스템을 위한 저장장치로 널리 사용되고 있다. 하지만, 아직까지는 NAND 플래시메모리의 단위 공간당 가격이 하드디스크에 비해 높은 실정이다. 따라서, 최근 노트북 컴퓨터 등에서 NAND 플래시메모리 기반의 SSD와 하드디스크를 동시에 사용하는 시도가 늘고 있다[1]. 이와 같은 환경에서 저장시스템의 목표는 이질적인 저장매체의 장점을 극대화하는 것이다. 즉, 스토리지 용량은 하드디스크처럼 사용하면서 전력소모나 접근 속도 측면에서는 플래시메모리처럼 사용하는 것이 그 목표이다.

서로 다른 물리적 특성을 가지는 두 저장매체의 장점을 최대한 살리기 위해서는 각 파일 블록이 어느 저장매체에 속한 것인지를 고려하는 효율적인 버퍼 캐쉬 관리 기법이 필수적이다. 특히, 모바일 시스템의 경우 배터리 제약조건 때문에 버퍼 캐쉬 관리 기법의 설계 시 매체별 전력소모율을 고려하는 것이 중요하다.

본 논문에서는 하드디스크와 NAND 플래시메모리 기반의 SSD가 함께 사용되는 모바일 시스템에서 전력 소모를 최소화하는 새로운 버퍼 캐쉬 관리 기법을 제안한다. 제안하는 기법은 캐싱된 블록의 가치를 평가하는 데 있어 각 저장 매체의 에너지 소모율, I/O 연산 유형(읽기/쓰기), 블록의 참조 가능성(최근성 및 빈도)을 모두 고려하면서도 효율적인 구현이 가능하다는 장점을 지닌다. 실제 파일 입출력 트레이스를 통한 시뮬레이션 실험을 통해 새롭게 제안된 기법이 LRU 알고리즘에 비해 스토리지의 전력 소모를 평균 18.0%, 최대 58.9%까지 줄일 수 있음을 보인다.

2. 절전형 버퍼 캐쉬 관리 기법

2.1 시스템 구조

본 논문의 저장 시스템 및 버퍼캐쉬 구조는 그림 1과 같다. NAND 플래시메모리 기반의 SSD와 하드디스크가 보조기억장치로 함께 사용되며 버퍼 캐쉬로 DRAM이 사용된다. 이와 같은 환경에서 전력 소모를 줄이기 위한 버퍼 캐쉬 관리 기법은 캐싱된 블록의 가치를 평

가하기 위해 아래의 3가지 조건을 고려해야 한다.

• 장치의 전력소모율

플래시메모리와 하드디스크는 매체의 접근을 위해 필요한 전력 소모가 서로 다르기 때문에, 버퍼캐쉬 관리 기법은 이를 고려해야 한다.

• 입출력 연산 유형

플래시메모리는 읽기 연산과 쓰기 연산에 드는 시간 및 전력 소모가 다르기 때문에 효율적인 버퍼 캐쉬 기법은 이를 고려해야 한다.

• 참조 가능성

전력 소모를 줄이기 위해서는 해당 블록이 버퍼 캐쉬에서 적중되어야 하므로 효율적인 버퍼 캐쉬 관리 기법은 캐싱된 블록의 참조 가능성을 잘 예측할 수 있어야 한다.

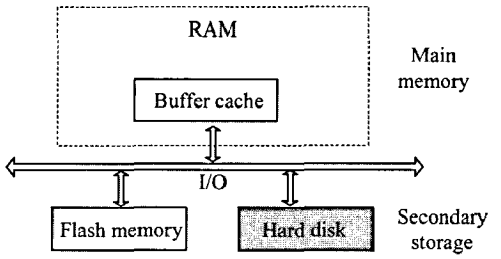


그림 1 본 논문의 스토리지 구성

2.2 알고리즘

본 절에서는 본 논문에서 제안하는 저전력 버퍼 캐쉬 관리 기법인 LBM(low-power buffer management) 기법에 대해 설명한다. LBM은 앞 절에서 소개한 세 가지 조건을 모두 고려해서 캐쉬 내의 블록을 평가한다.

첫째, 캐싱된 블록의 참조 가능성을 예측하기 위해 각 블록의 과거 참조 기록을 최근성(recency)과 참조빈도(frequency) 측면에서 모두 고려한다. 이는 최근 참조된 블록과 과거 참조 횟수가 많았던 블록이 가까운 미래에 다시 참조될 가능성이 높은 성질을 반영한 것이다. 대부분의 캐싱 환경에서 이와 같은 두 성질이 모두 나타나지만 이 둘의 상대적인 영향력은 환경에 따라 상이하다. 따라서, 효율적인 캐싱 알고리즘은 주어진 환경에 맞게 두 가지 특성을 적응적으로 반영할 수 있어야 한다. 이를 위해 본 논문에서는 LRFU 알고리즘에서 블록의 가치를 평가하기 위해 도입된 CRF(combined recency/frequency) 개념을 확장한다[2]. 각각의 캐싱된 블록은 그 블록의 참조 가능성을 예측하기 위한 CRF 값을 가지고 있다. 버퍼 캐쉬에 존재하는 동안 일어난 모든 과거 참조 기록이 CRF 값의 계산에 반영되며 각각의 참조에 의한 영향력은 가중치 함수인 $F(x)$에 의해 결정된다. 이 때 x는 참조 시점부터 현재까지 흐른 시간을 의미한다. LRFU 알고리즘에서와 달리 본 논문에서는 읽기 연산과 쓰기 연산을 구분하여 각 연산의 특성을 고려한 가중치를 곱하여 CRF 값을 계산한다. 예를 들어 그림 2와 같이 블록 i에 대한 읽기 연산이 시각 t_1과 t_2에 발생하고 쓰기 연산이 t_3에 발생했다고 하자. 그러면, 현재 시각 t_c에 $CRF(i)$ 값은 다음과 같이 계산된다.

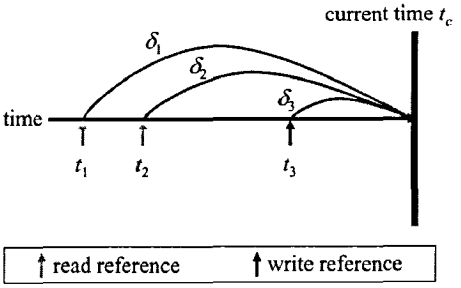


그림 2 LBM 기법의 CRF 값 계산

$$CRF(i) = r(i)\ F(\delta_1) + r(i)\ F(\delta_2) + w(i)\ F(\delta_3)$$

이 때, $\delta_1 = t_c - t_1$, $\delta_2 = t_c - t_2$, $\delta_3 = t_c - t_3$이고 $r(i)$와 $w(i)$는 각각 블록 i를 위한 읽기 가중치와 쓰기 가중치를 뜻한다.

$F(x)$는 감소함수로 정의하는 것이 적절한데 이는 최근 참조일수록 CRF 계산에 더 높은 영향력을 미치도록 해야 하기 때문이다. 본 논문에서는 LRFU 알고리즘에서 사용한 다음 함수를 그대로 적용하였다[7].

$$F(x)=(1/2)^{\lambda x}\ (0 \le \lambda \le 1).$$

λ 값이 0이면 $CRF(i)$는 과거의 읽기 및 쓰기 참조에 대한 가중합을 뜻하게 되며, λ 값이 증가함에 따라 최근 참조에 더 높은 가중치를 부여하게 된다.

CRF 값의 계산이 끝난 후에는 하드디스크와 플래시메모리 중 어느 저장매체에서 인출된 블록인지에 따라 매체 전력 소모율을 곱하여 블록의 최종적인 우선순위를 결정하게 된다. 예를 들어 하드디스크에서 인출된 블록의 경우 높은 우선순위를 부여하게 되는데 이는 하드디스크에서의 I/O 연산 비용이 플래시메모리의 경우보다 높기 때문이다.

즉, LBM 기법에서는 각 캐싱 블록 i마다 그들의 캐싱 우선순위를 결정하는 $Value(i)$를 계산하고 캐쉬 교체가 필요한 경우 $Value(i)$가 가장 작은 블록을 삭제한다. $Value(i)$의 계산식은 다음과 같다.

$$Value(i) = Weight_{dv}\ (i) \cdot CRF(i)$$

이 때, $Weight_{dv}(i)$는 블록 i를 저장매체로부터 접근하기 위해 소요되는 기본 전력 소모를 의미한다.

2.3 효율적인 구현 방법

본 절에서는 LBM 기법의 효율적인 구현 방법에 대해 설명한다. LBM 기법에서는 캐싱된 블록의 $Value$를

계산하기 위해 해당 블록의 모든 과거 참조 기록을 필요로 한다. 또한 I/O 연산 타입과 저장 매체 타입에 따라 두 종류의 가중치 값이 *Value*의 계산 시 곱해진다. 이와 같은 정보의 유지 및 관리는 LBM 기법이 실제 시스템에서 사용되기에 적합하지 않은 측면을 보여준다. 또한, 각 블록의 *Value*는 시간에 따라 변하기 때문에 시간이 흐름에 따라 모든 블록의 *Value*를 재계산해야 하는 오버헤드가 뒤따른다. 하지만, 본 논문에서는 이와 같은 문제점을 해결하고 LBM 기법의 효율적인 구현 방법이 존재함을 보여준다. LRFU 및 LUV 알고리즘에서도 비슷한 방식의 증명을 보였으며[2,3] 본 논문에서는 기존의 증명이 연산 타입이 구분되고 저장 매체 타입에 따른 가중치가 곱해지는 LBM 기법에서도 성립함을 보인 것이다.

성질 1. 블록 i의 n번째 참조 시점 t와 $(n+1)$번째 참조 시점 t' 시의 *Value*를 각각 $Value_t(i)$와 $Value_{t'}(i)$라 하자. 그러면 $Value_{t'}(i)$는 $Value_t(i)$로부터 다음과 같은 방식으로 유추될 수 있다.

Case 1. $(n+1)$번째 참조가 읽기 참조인 경우

$$Value_{t'}(i) = Value_t(i)\,F(\delta) + Weight_{dv}(i)\,r(i)$$

Case 2. $(n+1)$번째 참조가 쓰기 참조인 경우

$$Value_{t'}(i) = Value_t(i)\,F(\delta) + Weight_{dv}(i)\,w(i)$$

단, $\delta = t' - t$를 뜻한다.

증명. p와 q가 각각 과거의 읽기 참조 횟수와 쓰기 참조 횟수라고 하자. 그러면 $p+q=n$을 만족한다. δ_k는 n 번째 참조 시점과 k번째 읽기 참조 시점과의 시간 간격을 뜻하고$(1 \le k \le p)$, ρ_j는 n 번째 참조 시점과 j번째 쓰기 참조 시점과의 시간 간격을 뜻한다고 하자$(1 \le j \le q)$. 만약 $n+1$번째 참조가 읽기 참조라면 아래 식을 만족한다.

$$\begin{aligned}
&Value_{t'}(i)\\
&= Weight_{dv}(i)\,CRF_{t'}(i)\\
&= Weight_{dv}(i)\{r(i)\,F(\delta_1+\delta)+\ldots+r(i)\,F(\delta_p+\delta)+\\
&\quad w(i)\,F(\rho_1+\delta)+\ldots+w(i)\,F(\rho_q+\delta)+r(i)\,F(0)\}\\
&= Weight_{dv}(i)\{r(i)\,F(\delta_1)+\ldots+r(i)\,F(\delta_p)+w(i)\,F(\rho_1)\\
&\quad +\ldots+w(i)\,F(\rho_q)\}\,F(\delta)+Weight_{dv}(i)\,r(i)\\
&= Weight_{dv}(i)\,CRF_t(i)\,F(\delta)+Weight_{dv}(i)\,r(i)\\
&= Value_t(i)\,F(\delta)+Weight_{dv}(i)\,r(i)
\end{aligned}$$

비슷한 방식으로 $n+1$번째 참조가 쓰기 참조라면 아래의 식을 만족한다.

$$Value_{t'}(i) = Value_t(i)\,F(\delta)+Weight_{dv}(i)\,w(i)\ \square$$

성질 1은 $n+1$ 번째 참조 시점의 *Value* 계산이 n번째 참조 시점과 그 당시의 *Value* 및 $Weight_{dv}$에 의해 곧바로 얻어질 수 있음을 뜻한다. 이는 LBM 기법의 공간 복잡도가 블록 당 상수 바이트에 불과하여 공간 오버헤드 문제를 해소할 수 있음을 의미한다.

보조정리 1. 블록 i의 시각 t와 t' 시의 *Value*를 각각 $Value_t(i)$와 $Value_{t'}(i)$라 하자$(t' > t)$. 만약 t와 t' 사이에 블록 i에 대한 참조가 발생하지 않았다면 $Value_{t'}(i)$는 $Value_t(i)$로부터 다음과 같이 계산될 수 있다.

$$Value_{t'}(i) = Value_t(i) \cdot F(\delta)$$

단, $\delta = t' - t$를 뜻한다.

증명. 블록 i의 읽기 참조 횟수와 쓰기 참조 횟수를 각각 p와 q라 하고 δ_k와 ρ_j는 각각 k번째 읽기 참조 시점부터 t 시점까지 흐른 시간, j번째 쓰기 참조 시점부터 t 시점까지 흐른 시간 간격이라 하자. 그러면 다음 식이 성립한다.

$$\begin{aligned}
&Value_{t'}(i) = Weight_{dv}(i)\,CRF_{t'}(i)\\
&= Weight_{dv}(i)\{r(i)\,F(\delta_1+\delta)+\ldots+r(i)\,F(\delta_p+\delta)+\\
&\quad w(i)\,F(\rho_1+\delta)+\ldots+w(i)\,F(\rho_q+\delta)\}\\
&= Weight_{dv}(i)\{r(i)\,F(\delta_1)+\ldots+r(i)\,F(\delta_p)+\\
&\quad w(i)\,F(\rho_1)+\ldots+w(i)\,F(\rho_q)\}\,F(\delta)\\
&= Weight_{dv}(i)\,CRF_t(i)\,F(\delta) = Value_t(i)\,F(\delta)\ \square
\end{aligned}$$

보조정리 1은 블록 i의 최근에 계산된 *Value*와 계산 시점을 유지하고 있으면 임의의 시점에서의 *Value* 계산이 가능함을 의미한다.

성질 2. $Value_t(a) > Value_t(b)$를 만족하고 블록 a와 b가 모두 시각 t 이후 다시 참조되지 않았으면 임의의 시각 t' $(t' > t)$에 대해 $Value_{t'}(a) > Value_{t'}(b)$를 만족한다.

증명. $\delta = t' - t$라 하면 보조정리 1에 의해 다음 식을 만족한다.

$$\begin{aligned}
Value_{t'}(a) &= Value_t(a)F(\delta) > Value_t(b)F(\delta)\\
&= Value_{t'}(b)\\
&(\because F(\delta) > 0) \qquad \square
\end{aligned}$$

성질 2는 참조되지 않은 블록들 간에는 *Value*의 대소 관계가 변하지 않음을 뜻한다. 성질 1과 성질 2에 의해 LBM 기법은 힙 자료구조에 의한 효율적인 구현이 가능하므로 모든 캐쉬 연산은 $O(\log_2 n)$에 구현될 수 있다.

3. 성능 평가

본 논문에서는 성능 평가를 위해 시뮬레이션 실험을 수행하였다. SSD는 MTRON MSD-SATA3035-064 SSD 모델, 모바일 하드 디스크는 Hitachi Travelstar 5K100을 사용했으며 이들 장치의 특성은 표 1에 나타내었다[4]. 트레이스는 [5]에 소개된 파일 입출력 트레이스를 사용하였다. 각 트레이스는 2-5개의 프로세스를 동시에 실행시키면서 8KB 단위의 파일 블록 입출력 요청이 이루어진 기록을 담고 있다. 각 트레이스의 특징은 표 2에 나타내었다. 파일은 SSD와 하드 디스크에 2:8의 비율로 배치하였으며 이는 현 시장에서 하드 디스크가 SSD에 비해 가격 대비 큰 용량을 가지는 것에 근거하

표 1 실험에 사용된 스토리지의 특성[4]

	MTRON MSD-SATA 3035-064 SSD	Hitachi Travelstar 5K100
Price (US$/GB)	9.37	0.75
Active power dissipation (W)	0.8	4.2
Maximum transfer rate (MB/s)	90.5	38.8
Minimum transfer rate (MB/s)	90.5	22.1
Average random read time (ms)	0.1	16.4
Average random write time (ms)	6.8	16.4

표 2 실험에 사용한 트레이스

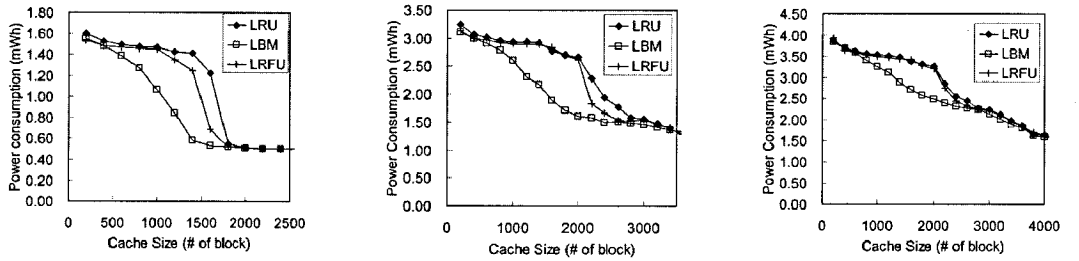
Trace	실행 프로세스	참조 횟수	참조 블록 수
Trace 1	cscope, cpp	15858	2606
Trace 2	cpp,cscope, postgres	32519	5456
Trace 3	cscope, cpp, postgres,mpeg, sdet	64062	7130

였다. LBM의 성능을 비교하기 위해 LRU와 LRFU 알고리즘을 함께 실험하였다. LBM과 LRFU의 파라미터 λ는 0.001로 하였고, LBM의 $Weight_{dv}$는 I/O 연산의 에너지 소모량을 고려하여 HDD는 10.8, SSD는 0.88로 하였다. 이는 표 1에 표시한 각 장치별 I/O 연산의 전력 소모량에 근거한 것이다.

그림 3은 I/O 연산으로 발생한 총 전력소모량을 보여준다. 그림에서 보는 것처럼 LBM의 전력 소모가 다른 알고리즘들에 비해 매우 작음을 알 수 있다. 특히, 캐쉬 용량이 워크로드 메모리 사용량의 20%에서 80% 구간일 때 전력 절감 효과가 가장 큰 것으로 나타났다. 80% 이상의 캐쉬 구간에서는 요청된 블록의 대부분을 수용할 만큼 캐쉬 용량이 큰 비현실적인 상황이어서 교체 발생이 거의 없으므로 세 알고리즘 모두 비슷한 결과를 나타내었다. 20% 미만의 캐쉬 구간일 때에는 짧은 재참조 간격(inter-reference gap)에 대해서만 캐쉬 적중을 얻을 수 있어 세 알고리즘이 큰 차이를 나타내지 않았다. 특히 실험에 사용된 세 트레이스 모두 재참조 중 상당량이 극히 짧은 재참조 간격(inter-refernce gap)을 가지며 이들은 어떤 알고리즘을 사용하더라도 대부분 캐쉬 적중을 얻을 수 있는 블록들에 해당된다. 예를 들어, 트레이스 3의 경우, 재참조간격이 46 블록 이내인 경우가 전체 재참조의 50%를 차지하며, 이와 유사하게 트레이스 1에서 160 블록의 재참조 간격 내에 40%, 트레이스 2에서는 170 블록의 재참조 간격 내에 발생하는 재참조가 전체 재참조의 18%를 차지한다.

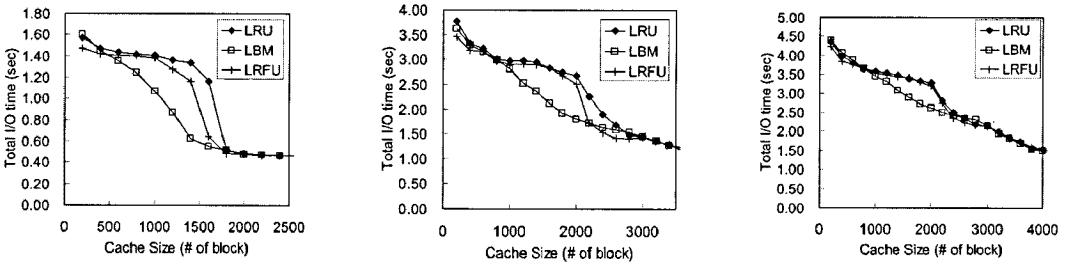
트레이스 1과 2는 긴 재참조 간격을 가지는 블록, 즉 루프 패턴의 블록들이 다수 존재하므로 LRFU와 LBM이 LRU에 비하여 좋은 성능을 나타내었다. 또한 LBM은 참조의 최근성과 빈도를 고려할 뿐 아니라 참조블록의 전력 소모량을 고려하므로 LRFU에 비해 전력 소모를 더욱 줄일 수 있었다.

그림 4는 캐쉬 용량에 따른 세 알고리즘의 총 I/O 시간을 보여주고 있다. 일부 구간을 제외하면 세 트레이스



(a) Trace 1 (b) Trace 2 (c) Trace 3

그림 3 알고리즘 별 전력 소모 비교



(a) Trace 1 (b) Trace 2 (c) Trace 3

그림 4 알고리즘 별 총 I/O 시간 비교

모두 총 전력 소모량과 비슷한 결과를 나타내었다. 일부 구간에서 LBM이 다른 알고리즘에 비해 성능이 저하되는 것은 LBM의 가중치가 소모되는 전력에 기반하여 저전력을 목적으로 설계되었기 때문이다.

4. 관련 연구

모바일 시스템에서 하드디스크와 플래시메모리를 함께 사용해서 스토리지를 구성하는 연구가 널리 이루어지고 있다. 이들 중 대부분은 플래시메모리를 하드디스크의 비휘발성 캐쉬로 사용하여 디스크의 스핀 다운 시간을 길게 하는 것을 주요 내용으로 하고 있다[1]. 본 연구는 하드디스크와 플래시메모리를 대등한 스토리지 계층으로 구성한다는 점에서 이들 기존 연구와 차별화된다.

본 논문과 같은 스토리지 환경에서 버퍼 캐쉬를 관리하는 서로 다른 두 가지 방법을 사용할 수 있다. 첫째는 각 스토리지 매체별로 버퍼 캐쉬 공간을 먼저 할당한 후, 할당된 영역을 독립적으로 관리하는 방법이다[6-8]. 이와 같은 방법을 사용할 경우 분할된 각 버퍼 캐쉬 공간에 대한 효율적인 관리가 가능하지만, 워크로드 변화에 따라 파티션의 크기를 동적으로 조절해야 한다는 문제점이 있다.

두 번째 방법은 버퍼 캐쉬를 각 저장 매체별로 미리 할당하지 않고 하나의 풀로 관리하는 방법이다. 본 논문은 두 번째 방법을 사용했는데 이는 버퍼 캐쉬의 관리 오버헤드가 적을 뿐 아니라 동적인 파라미터 튜닝이 필요없기 때문이다. 이와 같은 방식의 한 가지 문제점은 캐싱된 블록들의 평가를 위해 재참조 가능성 뿐 아니라 해당 블록의 원본이 위치한 저장 매체의 I/O 특성을 고려해야 하므로 캐쉬 교체 알고리즘의 구현이 복잡해진다는 점이다. 그러나, 본 논문에서 제안하는 기법은 몇 가지 성질을 만족하기 때문에 효율적인 구현이 가능하다는 것을 보였다.

두 가지 방법에 대한 효과를 실험을 통해 비교할 수 있으나 본 논문에서는 파티션에 기반한 방법과의 성능 비교 결과는 담지 않았다. 이는 파티션 기반의 기존 알고리즘들이 본 논문과는 상이한 성능 척도 및 시스템 아키텍처를 사용하고 있기 때문이다. 예를 들어 [6-8] 모두 전력 소모 절감보다는 성능 향상에 초점을 맞추고 있으며, [7]은 성능 특성이 다른 하드디스크만을 스토리지로 사용하고 [6]은 플래시메모리만을 스토리지로 사용하고 있다. 본 논문의 후속 연구에서는 파티션에 기반한 저전력 버퍼 캐쉬와 LBM의 비교 연구를 수행하고자 한다. 이에 대한 선행 실험으로 파티션의 크기를 매체별 전력 소모량에 근거해서 결정하는 기본적인 방법을 사용한 결과 LBM에 비해 전력 소모량이 다소 높은 것으로 나타났다. 하지만, 파티션의 크기를 동적으로 잘 조절할 경우 LBM과 유사하거나 좋은 성능을 나타내는 경우도 있어 이에 대해서는 후속 연구가 필요할 것으로 생각된다.

5. 결 론

본 논문은 HDD와 SSD를 저장장치로 사용하는 시스템을 위한 저전력 버퍼 캐쉬 관리 기법을 제안하여 모바일 시스템의 배터리 제약을 완화하였다. 한편, 본 논문은 휘발성 RAM을 버퍼 캐쉬로 사용하므로 pdflush 데몬에 의한 주기적인 flush에 의해 약한 일관성 유지만 보장할 뿐 전원 오류시 더티 블록에 대한 일관성을 100% 보장하지는 못한다. 향후에는 바이트 단위로 접근이 가능하며 비휘발성의 성질을 가지는 PCM, MRAM 등을 더티 블록을 위한 버퍼캐쉬로 이용하여 이와 같은 일관성 문제를 100% 해결하는 버퍼 캐싱 기법으로 확장하고자 한다.

참 고 문 헌

[1] F. Chen, S. Jiang, and X. Zhang, "SmartSaver: Turning Flash Drive into a Disk Energy Saver for Mobile Computers," *Proc. Int'l Symp. on Low Power Electronics and Design*, pp.412-417, 2006.

[2] D. Lee, J. Choi, J.H. Kim, S.H. Noh, S.L. Min, Y. Cho, C.S. Kim, "LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies," *IEEE Trans. Computers*, vol.50, no.12, pp.1352-1361, 2001.

[3] H. Bahn, S. H. Noh, S. L. Min, and K. Koh, "Efficient Replacement of Nonuniform Objects in Web Caches," *IEEE Computer*, vol.35, no.6, pp.65-73, 2002.

[4] Storagereview.com, http://www.storagereview.com/

[5] J.M. Kim, J. Choi, J. Kim, S.H. Noh, S.L. Min, Y. Cho, C.S. Kim, "A Low-Overhead High-Performance Unified Buffer Management Scheme that Exploits Sequential and Looping References," *Proc. Int'l Symp. Operating System Design & Implementation*, 2000.

[6] J. Park, H. Lee, S. Hyun, K. Koh, and H. Bahn, "A Cost-aware Page Replacement Algorithm for NAND Flash Based Mobile Embedded Systems," *Proc. ACM Int'l Conf. on Embedded Software (EMSOFT)*, 2009.

[7] B. Forney, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Storage-Aware Caching: Revisiting Caching for Heterogeneous Storage Systems," *Proc. USENIX Symp. File and Storage Tech. (FAST)*, 2002.

[8] Y.-J. Kim and J. Kim, "DAC: A Device-Aware Cache Management Algorithm for Heterogeneous Mobile Storage Systems," *IEICE Trans. Information and Systems*, 2008.