논문 2012-07-16

이기종 저장장치를 위한 제거 비용 평가 기반 캐시 관리 기법

(A Cache Management Technique Based on Eviction Cost Estimation for Heterogeneous Storage Devices)

박 세 진*, 박 찬 익
(Sejin Park, Chanik Park)

Abstract : The objective of cache is to reduce I/O access of physical storage device so that user accesses their data faster. Traditionally, the most important metric to measure the performance of cache is hitratio. Thus, when the cache maintains hitratio high, it is regarded as a good cache replacement policy. However, the cache miss latency is different when the storages are heterogeneous. Though the cache hitratio is high, if the cache often misses with low performance disk, then the user experiences low performance. To address this problem we proposed eviction cost estimation based cache management. In our result, the eviction cost estimation based cache management has 10~30% throughput improvement compared with LRU cache management.

Keywords : Cache, Heterogeneous, Storage, Eviction, Estimation.

Ⅰ. 서 론

로컬에 있는 저장장치는 다양한 저장장치와 [1, 2, 3] 혼용이 되어 많이 쓰인다. 기존의 네트워크 기반 저장장치들을 비롯해, 최근에 많이 나오는 클라우드 기반 스토리지 서비스 [4], 그리고 Solid State Drive (SSD) 등의 고성능 저장장치들이 많이 보급 되고, 이러한 저장장치는 기존의 하드 디스크와 혼용되어 사용되고 있다.

일반적으로 성능이 다른 저장장치가 있을 경우, 자주 사용하는 데이터를 고 성능 저장장치에 배치시키는 것이 유리한데, HDD와 SSD가 혼용되어 사용되는 시스템을 예로 들면 데이터 재배치에 크게 다음 두 가지의 방법이 있다.

1. 사용자가 데이터 직접 접근 빈도수를 예상하여 수작업으로 HDD 와 SSD에 데이터를 재배치시키는 방법
2. 데이터 재배치 소프트웨어가 사용자의 데이터 사용빈도를 측정 후 자동으로 배치시키는 방법

1번 의 경우, 사용자가 직접 접근 빈도수를 측정하기가 쉽지 않으며, 저장장치에 대한 지식이 없는 사람들은 직접 할 수 없다. 2번의 경우, 사용자의 의도와 다르게 데이터들의 재배치가 일어나게 되어, 추후 데이터 검색에 혼란을 야기될 수 있다.

본 논문에서는 이러한 이기종 저장장치의 특성을 고려한 고성능 캐시 정책을 설명한다. 제안하는 캐시를 사용하면 추가적인 데이터 재배치 없이, 고성능 효과를 낼 수 있게 된다.

구체적으로 본 논문에서는 SSD와 HDD등이 혼

* 교신저자(Corresponding Author)
논문접수 : 2012. 01. 31., 수정일 : 2012. 02. 25.,
채택확정 : 2012. 03. 23.
박찬익, 박세진 : 포항공과대학교 컴퓨터공학과
※ 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원 (No. 2011 - 0016972) 및 지식경제부 및 정보통신 산업 진흥원의 “IT명품 인재 양성 사업”의 (C1515 - 1121 - 0003) 연구결과로 수행 되었음"

재해 있는 이기종 저장장치에서는 각 장치별 접근 시간의 차이가 많이 나기 때문에, 캐시의 hitratio 보다 miss 시 발생하는 저장장치의 접근 비용이 더 중요하다는 사실을 설명하고, 이를 바탕으로 각 저장장치의 특징을 고려한 새로운 캐시정책을 제안한다.

2장에서는 기반 지식을 설명하며 3장에서는 설계를 설명하고, 4장에서는 제안하는 캐시 성능 평가를 한다. 그리고 5장에서 결론을 맺으며 본 논문을 정리한다.

II. 기반 지식

캐시는 저장장치에 접근하는 입출력 횟수를 줄여서, 사용자가 더 빠르게 데이터에 접근 할 수 있게 해 준다. 전체 시스템의 병목현상을 일으키는 저장장치 접근을 효율적으로 처리하는 캐시는 서버, 데스크탑 시스템 뿐 만 아니라, 임베디드 시스템에서도 매우 중요하다. 이러한 캐시의 성능은 캐시 적중률을 통해 대변되는데, 일반적으로 다양한 워크로드에 대해서 높은 캐시 적중률을 유지시킬 수 있는 캐시가 좋은 캐시이다.

그러나 각 저장장치의 성능이 다른 이기종 저장장치 환경이라면 단순히 캐시 적중률만으로는 캐시의 성능을 평가할 수 없다. 만약, 전체 캐시 적중률은 높지만, 성능이 낮은 저장장치로의 접근에 대한 캐시 적중률이 매우 낮아, 해당 저장 장치에 대한 캐싱 효과를 보지 못하고, 직접 접근이 많이 일어난다면, 사용자가 느끼는 성능 하락은 심각해 질 수 있다.

그림 1은 SSD, HDD, Network Storage 가 연결되어 있는 일반적인 이기종 저장장치 환경을 나타내고 있다. 자주 접근되는 데이터는 버퍼캐시에 유지 되고 있으므로, 실제 데이터에 접근되는 시간이 Cache Hit latency로 동일하지만, Cache Miss 의 경우, 실제 데이터가 위치하고 있는 스토리지에 접근해야 하므로, 해당 스토리지의 접근 Latency가 필요하게 된다. 즉, 고성능의 SSD 와 원거리의 Network Storage의 접근 Latency가 차이가 많이 나기 때문에 스토리지별 캐시 Miss Latency 가 달라지게 되는 것이다. 이것은 기존 캐시 알고리즘의 가정을 벗어나기 때문에, Miss Latency를 고려한 새로운 캐시 정책이 필요하게 된다. 기존의 버퍼캐시 알고리즘은 LRU

이와 비슷한 연구로 VDF [5] 가 있다. 이는 디스크 어레이 시스템에서 특정 디스크가 고장 난 경우 해당 디스크에 대한 접근 페널티를 다르게 하는 캐시 정책을 지원하고 있다. 이 연구의 경우, 디스크 복구 시간 및 접근 시간을 줄였으나, 별도의 두 개의 운영모드(일반모드, 고장상태 모드) 가 있어, 운영 모드 변경 시 오버헤드가 있으며, 고장 모드에서만 제시한 알고리즘이 적용된다는 한계점을 가지고 있다.

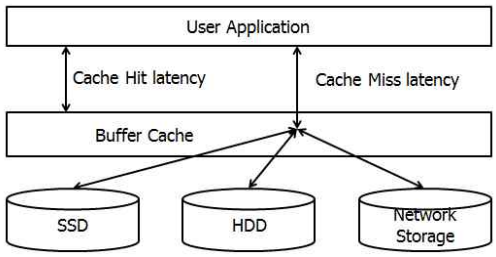


그림 1. 이기종 저장장치 환경
Fig. 1. Heterogeneous storage environment

III. 설 계

1. 이기종 저장장치 성능 척도

앞서 살펴본 것처럼 기존의 캐시정책은 하부 저장장치의 접근 시간이 동일하다는 가정 하에 만들어진 것으로, 각 장치별 접근시간이 다른 이기종 저장장치 시스템의 특성을 반영하지 못한다. Miss 가 매번 느린 저장장치에서 일어날 경우, 이 느린 접근 시간 때문에, 이기종 환경에서는 캐시적중률이 높더라도, Throughput 이 떨어지는 경우가 발생하게 된다. 따라서 Throughput을 높게 유지 시켜주는 새로운 캐시 관리 기법이 필요하다.

저장 장치의 성능은 크게 미디어 자체의 특성과 워크로드의 특성에 따라 결정되는데, SSD 와 HDD 의 예를 들면, 두 드라이브는 미디어적으로 입출력 성능이 다르며, 각 드라이브는 읽기, 쓰기, 순차접근, 임의 접근등 워크로드 특성에 따라 다양한 성능이 나오게 된다. 본 연구에서는 이러한 다양한 성능 나타내는 척도로 Throughput을 사용하며, 이러한 특성을 반영하기 위해 제거비용 그룹이라는 개념을 도입한다.

표 1은 SSD 와 HDD에서 미디어 특성과 워크로드 특성에 따라서 4개의 그룹을 관리하는 경우를 나타낸 것 이다. 각 그룹별 Throughput 이 다르기 때문에, 데이터가 캐시에서 제거될 때의 비용을 다르게

처리해야 효과적인 캐시운용이 가능한데, 이를 위해 각 그룹별 제거 비용값이 별도로 존재하며, 각 그룹의 제거 비용은 Throughput에 반비례한다. 가장 높은 Throughput을 가진 그룹의 제거비용을 1로 두고, 각 그룹의 제거비용은 해당 그룹의 Throughput에 반비례한 값을 가지게 된다. 표 1의 경우, Throughput이 160MB/s 인 그룹 1의 제거 비용이 1이 되고, Throughput이 40 MB/s인 그룹 4의 제거 비용은 4배가 느리기 때문에 4가 된다.

표 1. 제거 비용 그룹 예제
Table 1. Example of eviction cost groups

그룹	특징	Throughput	제거비용
1	SSD, 순차읽기	160 MB/s	1
2	SSD, 임의읽기	120 MB/s	4/3
3	HDD, 순차읽기	80 MB/s	2
4	HDD, 임의읽기	40 MB/s	4

2. 캐시 관리 기법

표 1에서 나타난 것처럼 다양한 이기종 저장장치들은 미디어 특징과 워크로드 특징에 따라 산출되는 Throughput을 기준으로 캐시 제거 비용 그룹 단위로 나누어 질 수 있다. 본 연구에서는 이 자료구조를 효과적으로 관리하기 위해서 복수개의 LRU queue와 하나의 제거 리스트를 사용한다.

그림 2는 제거 그룹이 4개가 있는 시스템에 대한 캐시 관리를 위한 LRU queue 구조이다.

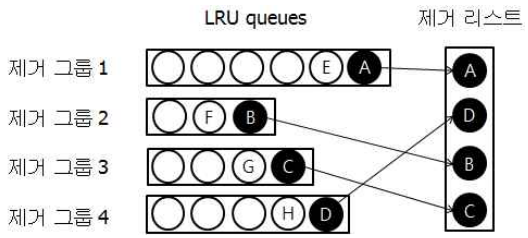


그림 2. 제거 비용 그룹 관리 자료 구조
Fig. 2. Data structure for eviction cost group

캐시는 각 제거 비용 그룹마다 독립적인 LRU queue를 유지하여, 해당 그룹에 대한 접근이 일어나면, 순차적으로 LRU queue 에 저장 된다. 이때 각 엔트리는 queue 에 삽입될 때의 time stamp 값을 저장하여 추후 캐시 교체 작업에 사용된다. 각 그룹이 관리하고 있는 LRU queue 의 전체 크기는 시스템의 캐시크기 이며, 캐시 교체에 필요한 연산을 줄이기 위해 하나의 별도 제거 리스트를 유지한다. 이 제거리스트는 각 LRU queue 의 마지막 엔트리들을 유지하고 있으며, 캐시 교체시, 제거 리스트에 있는 엔트리중 다음 비교 값이 가장 낮은 엔트리를 제거한다.

*비교 값 = 엔트리의 timestamp * 해당 엔트리가 속한 그룹의 제거비용*

따라서 본 연구에서 제안하는 캐시 알고리즘은 다음과 같다.

정의

RBi = 요청된 블록
lru_q[i] = 그룹 i의 LRU queue
evict_cost[i] = 그룹 i의 제거 비용
g_timestamp = 전역 timestamp, 초기값은 0

알고리즘

```
If ( RBi 가 캐시에 존재하면 ) // cache hit
   RBi를 lru_q[RBi.group]의 top 으로 이동;
}
else {                          // cache miss
   if (캐시가 가득찼으면) {
      for 제거 리스트에 있는 모든 블록 Bi {
          calc_cost[i] = Bi.timestamp *
          evict_cost[Bi.group];
      }
      calc_cost[i] 가 최소값인 블록 Bi 제거
      하고, 해당 블록이 속한 그룹의 lru_q의
      가장 마지막 엔트리를 제거 리스트에 삽
      입;

   }
   RBi.group = 저장 미디어 및 워크로드에 따
               른 그룹 분류
   RBi.timestamp = g_timestamp++;
   lru_q[RBi.group].push(RBi);

}
```

위 알고리즘에서 RBi.group 값은 미리 정의된 그룹에 의해 분류가 된다. 예를 들어(표 1 의 경우) SSD 와 HDD 가 공존하는 시스템에, 순차 읽기와 임의 읽기를 워크로드 특성으로 분류한다면. 이 경

우, 그룹은 4개가 발생되며, RBi 의 특성에 따라 그룹을 정해줄 수 있게 된다.

각 LRU queue는 시간적 순서대로 엔트리가 존재하기 때문에 적어도, LRU queue 내에서는 접근 순서가 지켜지고 있다. 즉, 각 LRU queue의 마지막 엔트리들만 비교하는 것이 결과적으로 전체 캐시를 비교하는 것과 동일한 효과를 가지게 된다. 또한 본 연구에서 제시하는 캐시 알고리즘은 교체 엔트리 검색 시간이 항상 “전체 제거 그룹의 개수“의 상수 값을 가지게 되므로 O(1)의 시간 복잡도를 가진다.

IV. 평 가

제안한 캐시관리기법의 성능을 평가하기 위해 그림 3과 같은 시뮬레이션 환경을 구축했다. 시뮬레이터의 워크로드 재생기는 지정된 워크로드와 동일한 I/O를 I/O 분배기로 계속 내려주며, I/O 분배기는 실험을 위해 SSD 와 HDD 에 대한 접근 비율을 지정된 파라미터에 따라 분배 해준다. 본 실험에서는 SSD : HDD 접근 비율을 0:100 / 20:80 / 40:60 / 60:40 / 80:20 / 100:0 과 같이 다양하게 셋업 하였다.

실험의 단순화를 위해 이기종 저장장치는 SSD 과 HDD 두 가지를 설정하였으며, 캐시 크기는 1024 MB 로 설정하였다. 또한, 제거 비용 그룹은 SSD와 HDD 두 개의 그룹만을 유지하며, SSD 의 throughput은 150 MB/s, HDD의 throughput은 50 MB/s 으로 각 그룹의 제거비용은 SSD = 1과 HDD = 3으로 설정하였다. 이는 SSD 의 throughput 이 3배 더 큰 것을 의미한다.

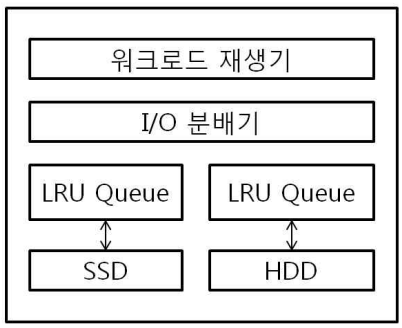
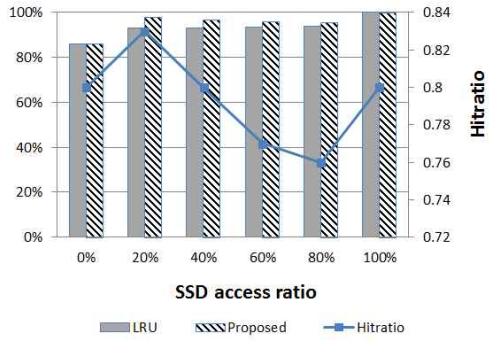
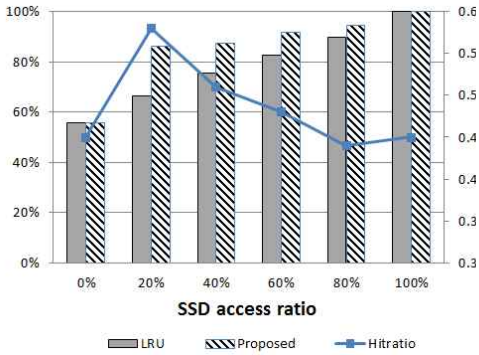


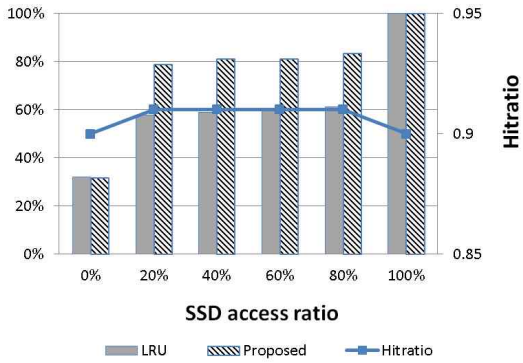
그림 3. 이기종 캐시 평가용 시뮬레이터
Figure 3. Heterogeneous cache simulator



(a) MSR usr_0 워크로드
(a) MSR usr_0 workload



(b) MSR proj_0 워크로드
(b) MSR proj_0 workload



(c) MSR usr_0 워크로드
(c) MSR usr_0 workload

그림 4. 다양한 워크로드를 이용한 실험 결과
Fig. 4. Experimental result using various workloads

본 논문에서 사용한 실험 워크로드는 MSR Cambridge 의 usr_0와 proj_0 [6] 워크로드를 사

용하였다. 1024 MB Cache size와 2048 MB cache size 에서 usr_0 워크로드의 순수 LRU 적용시 hitratio 는 각각 0.8, 0.9 이며, proj_0의 순수 LRU 적용시 hitratio 는 1024 MB 크기에서 0.45 이다.

그림 4(a)는 MSR usr_0 워크로드에 대한 결과이며, (b)는 proj_0 에 대한 실험 결과이다. 왼쪽 세로축은 상대적인 throughput 이며, SSD에 100% 비율로 접근할 때의 throughput 이 100% 이다. 오른쪽 세로축은 Hitratio 이다. 제안하는 알고리즘을 적용하였을 경우 항상 LRU 보다 좋은 성능이 나오는 것을 알 수 있으며, 특히 그림 4(a)에서 SSD : HDD 접근비율이 20:80 일 때 30% 정도의 가장 큰 성능 향상을 나타내고 있다.

그림 4(b)의 proj_0의 경우 SSD의 접근 비율구간이 20%에서 80% 까지 올라가면서 Hitratio 는 떨어지고 있지만, 오히려, throughput은 점점 더 증가하는 것을 볼 수 있다. 이는 이기종 저장장치에서 캐시를 유지하기 위해서는 기존의 캐시 성능 척도인 캐시 적중률만으로는 부족하다는 것을 보여주고 있으며, 동시에 제안하고 있는 캐시 알고리즘이 해당 문제를 효과적으로 지원하고 있음을 보여주고 있다.

그림 4(c) 는 MSR usr_0 워크로드를 2048 MB 캐시 크기를 가질 때의 결과 값이다. 전체적으로 SSD 와 HDD 에 대한 접근이 동시에 일어나는 구간(20% ~ 80%)에서 LRU 보다 약 25% 정도 좋은 Throughput을 보여주고 있다.

V. 결 론

본 연구는 기존의 캐시 관리 기법에서 고려하고 있지 않은 이기종 저장 장치에 대한 효과적인 캐시 관리 방법에 대한 내용을 설명하고 있다. 기존의 캐시 알고리즘은 하부 스토리지 시스템의 접근 시간이 동일하다고 가정하고 디자인되어 있으나, 이기종 저장장치 시스템에서는 실제 각 저장장치별 접근 시간이 다르기 때문에, 기존 캐시 알고리즘이 제대로 작동하지 않는다.

이를 위해 본 연구에서는 이기종 저장장치를 효과적으로 지원하기 위해 저장 장치들을 다양한 특성에 따라 제거 비용 그룹이라는 형태로 구별하며, 별도의 LRU queue로 유지하는 캐시 관리 기법을 제안하였다.

실험 결과 워크로드에 따라 최대 30% 까지 성능향상이 있음을 보이고 있다. 본 연구에서는 시뮬레이션 기반의 실험 결과를 도출하고 있지만, 실제 시스템으로의 적용이 필요하다.

참고문헌

[1] B. Pawlowski, S. Shepler, C. Beame, B. Callaghan, M. Eisler, D. Noveck, D. Robinson, and R. Thurlow. The NFS Version 4 Protocol. http://www.connectathon.org/talks97/index.html, 1997.

[2] IETF, Internet Small Computer Systems Interface (iSCSI), http://www.ietf.org/rfc/rfc3720.txt.

[3] CIFS, Common Internet File System, www.samba.org/cifs

[4] Amazon Simple Storage Service, http://aws.amazon.com/s3/

[5] S. Wan, Q. Cao, J. Huang, S. Li, X. Li, S. Zhan, L. Yu, C. Xie, X. He, “Victim disk first: an asymmetric cache to boost the performance of disk arrays under faulty conditions,” Proceedings on the 2011 USENIX Annual Technical Conference, pp.173-186, 2011.

[6] MSR Cambridge workloads www.snia.org

저 자 소 개

박 세 진

2007년 금오공과대학교 소프트웨어공학과(학사).
2007년~현재 포항공과대학교 컴퓨터공학과 통합과정.
관심분야: 운영체제, 가상머신, 임베디드 시스템
E-mail: baksejin@postech.ac.kr

박 찬 익

1983년 서울대학교 전자공학과(학사).
1985년 한국과학기술원 컴퓨터공학과(석사).
1988년 한국과학기술원 컴퓨터공학과(박사).
1989년~현재 포항공과대학교 정교수
관심분야: Storage systems, System security, Cloud computing & Virtualization, Embedded Linux
E-mail: cipark@postech.ac.kr