오염 버퍼를 적용한 집합 연상 페이지 캐시 기법

안득현, 김지홍, 엄영익
성균관대학교 정보통신대학
e-mail : {elise, jjilong, yieom}@skku.edu

A Enhanced Set-Associative Page Cache Scheme using Pollute Buffer

Deukhyeon An, Jeehong Kim, and Young Ik Eom
College of Information and Communication Eng., Sungkyunkwan University

요 약

큰 데이터 트래픽을 일으키는 I/O 작업을 수행할 경우에 많은 디스크 접근과 데이터 처리가 발생하며 이는 컴퓨팅 성능의 하락을 일으킨다. 이를 위해 메모리와 디스크 사이에 버퍼 역할을 하는 페이지 캐시 기법이 사용된다. 그러나 LRU 를 사용하는 페이지 캐시의 특성상, 많은 양의 데이터가 한번만 접근되고 다시 사용되지 않는다면 성능상의 큰 효과가 없다. 본 논문에서는 집합 연상 페이지 캐시에 오염 버퍼를 둠으로써, 재사용되지 못하고 페이지 캐시의 크기만 커지는 현상을 최소화 시켜 I/O 성능을 개선시킬 수 있는 방법을 제안한다.

1. 서론

하드디스크와 같은 2 차 저장장치에 메모리의 정보를 저장하고자 할 때에는 디스크의 처리 속도 때문에 상당한 접근시간이 요구된다. 이를 개선하기 위해 여러 운영체제들은 메모리와 디스크 사이에 캐시 역할을 하는 버퍼를 두고 있으며, 특히 리눅스에서는 페이지 캐시(Page Cache)기법을 이용한다 [1]. 그러나 DB 와 같이 많은 데이터의 처리가 디스크에 요구될 때, 페이지 캐시는 메모리의 공간을 차지함과 동시에 CPU 캐시의 오염 현상(Cache Pollution)을 야기하여 컴퓨팅 성능을 하락시키는 요소가 된다 [2]. 이전의 연구들에선 이와 비슷한 문제를 해결하기 위해 페이지캐시와 캐시 오염에 대해 다루고 있다 [3][4][5]. 본 논문에서는 집합 연상(Set-Associative) 페이지 캐시 기법을 이용하여 캐시 오염을 일으킬만한 스레드에 페이지 캐시를 따로 격리하여 처리함으로 앞서 얘기한 문제를 해결하고자 하는 기법을 제안하였다.

본 논문의 나머지 장은 다음과 같이 구성되어 있다. 2 장에서는 본 논문의 배경과 동기 및 기존의 연구를 간단히 소개한다. 3 장에선 오염 버퍼를 둔 집합 연상 페이지 캐시 기법에 대하여 기술한다. 마지막으로, 4 장에서는 본 논문에 대한 결론과 향후 연구 계획에 대하여 토론하며 마친다.

이 논문은 2012 년도 정부(교육과학기술부)의 재원으로 한국연구재단 기초연구사업의 지원을 받아 수행된 연구임(2010-0022570)

2. 배경 및 관련연구

2.1 I/O 접근에 의한 처리 지연

어떤 스레드가 작업을 마치기 위해 아래 표 1 과 같이 CPU 시간이 필요하며, 수행하는 동안 각 해당 I/O 접근을 하는 작업이 있다고 가정하자. 현재 서버에 쓰이는 15000rpm 의 HDD 중 최신 모델이 한 번의 I/O 연산당(IOPS) 약 3ms 의 I/O 처리시간을 필요로 한다. 설명의 편의를 위해 한 번의 I/O 연산이 디스크의 한 블록을 접근한다고 하면 스레드 A 는 작업을 마치기 위해 적어도 약 600ms, 스레드 B 는 1200ms 의 I/O 처리시간이 요구된다. 따라서 I/O 연산이 많으면 많을수록, 처리 시간이 CPU 시간과 비교하여 상당한 차이가 나는 것을 알 수 있다 [6].

<표 1> 스레드의 요구 CPU 시간 및 I/O 접근빈도

Thread	CPU	I/O rate
A	400ms	2No./ms
B	400ms	1No./ms

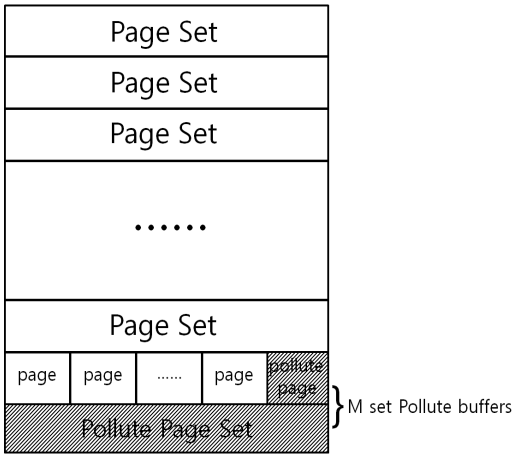
현재 인텔 네할렘 (Nehalem)마이크로프로세서 구조에서는 메모리에 접근하는데 일반적으로 약 60ns 의 처리시간이 소요된다 [7]. 이것은 I/O 처리시간인 약 3ms 과 비교하면 20μs 배라는 엄청난 차이를 보여준다. DMA 를 이용한다 하더라도 CPU 와 DMA 간의 추가적인 부담이 있으며 전체 I/O 처리시간은 해결할 수가 없다. 페이지 캐시는 이러한 부담을 해소시킬 수 있는 매우 유용한 역할을 하고 있어 잘 다룰 필요가 있다.

2.2 페이지 캐시

리눅스에는 디스크와 메모리간의 버퍼 역할을 위한 페이지 캐시라는 디스크 캐시가 구현되어 있다 [1]. 어떤 작업에서 디스크에 대한 접근이 필요할 때 해당 물리적 블록의 값을 가지고 있는 메모리에 대신 접근함으로써 I/O 연산을 최소화 시키는 기법이다. 페이지 캐시의 크기는 가용메모리에 따라 동적으로 변하며, 가용메모리가 부족할 경우 페이지 캐시 사이즈를 줄이거나 LRU(Least Recently Used)에 의해 캐시의 내용을 비운다(Cache Eviction). 하지만 많은 양의 데이터를 한번만 접근하고 다시 사용하지 않는다면 LRU 에 의한 이득을 보기는 어렵다. 또한 이로 인해 가용메모리가 부족해지면 캐시된 데이터를 재사용하지 못하고 다시 페이지 캐시의 사이즈를 줄이므로 쓸데없는 컴퓨팅 비용이 발생한다. 이러한 방식으로 페이지 캐시가 할당되고 비워지기 때문에 큰 I/O 트래픽을 일으키며 디스크에 접근하는 작업일 경우에는 이러한 문제를 해결할 수 있는 방법이 필요하다.

3. 오염 버퍼를 둔 집합 연상 페이지 캐시 기법

페이지 캐시에 대한 이전의 연구들 [4][5] 중에 특히 Da Zheng 외 2 명은 아래의 그림 1 과 같이 작은 N 개의 여러 페이지를 모아 하나의 집합으로 구성한다. 오염 버퍼 [3]를 이용한 집합 연상 페이지 캐시 기법은 이러한 집합 연상 구조를 가지고 있으며, 어떠한 스레드가 디스크로부터 큰 데이터 트래픽을 일으키는 I/O 작업을 수행할 경우에 동적으로 오염 페이지 부분으로 사상한다. 따라서 캐시된 데이터를 재사용하지 못하고 페이지 캐시의 크기만 커지는 현상이 발생하는 것을 해결할 수 있다.



(그림 1) 오염 버퍼를 둔 집합 연상 페이지 캐시 구조도

리눅스의 커널 내부에선 스레드의 I/O 트래픽이 발생하면 /proc 경로 아래에 실시간으로 기록한다 [8]. 이를 이용하여 현재 실행중인 스레드의 I/O 트래픽이 일정 값을 넘으면 오염을 일으킬 수 있는 스레드로 간주하여 오염 페이지 캐시로 사상시켜 줄 수 있다. 페이지 구조에 대한 대략적인 의사코드를 아래 그림 2 와 같이 나타내었다.

```
struct page_set {
        unsigned long flags;
        spinlock_t lock;
        ...
        int nrpages;
        page pages[NUMBER_OF_PAGES];
        int pollute = 0;
}
```

```
struct page {
        unsigned long flags;
        struct address_space *mapping;
        int pollute = 0;
        ...
        struct list_head lru;
        void *freelist
        struct kmem_cache *slab
        void *shadow;
        }
```

(그림 2) 오염 버퍼를 둔 집합 연상 페이지 캐시의 의사코드

4. 결론 및 향후 연구

본 논문에선 대용량의 데이터를 디스크에서 처리할 때 컴퓨팅 성능에 어떠한 영향을 주는지 알아보았다. 또한 이 때문에 페이지 캐시가 버퍼 역할을 하지만 또 다른 문제점이 발생한다는 것을 알았다. 이를 해결하기 위해 오염 버퍼를 둔 집합 연상 페이지 캐시 기법을 적용하여, 재사용되지 못하고 페이지 캐시의 크기만 커지는 현상을 최소화 시킬 수 있는 방법을 알아보았다. 향후 본 논문의 제안 기법을 실제 리눅스 커널에서 구현하고 적용하여 평가할 계획이다.

참고문헌

[1] Love, L.: Linux Kernel Development - Third Edition. Addison Wesley, (2010)

[2] Ding, X., Wang, K., Zhang, X.: SRM-buffer: An OS buffer management technique to pre-vent last level cache from thrashing in multicores. In: 6th ACM European Conference on Computer Systems, pp. 243-256. (2011)

[3] Soares, L., Tam, D., Stumm, M.: Reducing the Harmful Effects of Last-Level Cache Polluters with an OS-Level, Software-Only Pollute Buffer. In: IEEE MICROarchitecture, pp. 258-269. (2008)

[4] Zheng D., Burns, R., Szalay, A., S.: A Parallel Page Cache: IOPS and Caching for Multicore Systems. In: 4th USENIX conference on Hot Topics in Storage and File Systems, pp. 5-5. (2012)

[5] Jiang, S., Chen, F., Zhang, X.: CLOCK-Pro: An effective improvement of the CLOCK replacement. In: USENIX Annual Technical Conference, pp. 35-35. (2005)

[6] Qin, X., Jiang, H., Zhu, Y., Swanson, D., R.: Boosting Performance for I/O-Intensive Workload by Preemptive Job Migrations in a Cluster System . In: 15th Symposium on Computer Architecture and High Performance Computing, pp. 235. (2003)

[7] Molka, D., Hackenberg, D., Schöne, R., Müller, M.S.: Memory Performance and Coheren-cy Effects on an Intel Nehalem Multiprocessor System. In: 18th IEEE Parallel Architec-tures and Compilation Techniques, pp. 261-270. (2009)

[8] The Linux Kernel Archives: THE /proc FILESYSTEM, http://www.kernel.org/doc/Documentation/filesystems/proc.txt