저전력 버퍼 캐시[1]

이민[O] 서의성 이준원
한국과학기술원
{mlee[O], ses, joon}@camars.kaist.ac.kr

Power Aware Buffer Cache

Min Lee[O], Euiseong Seo, Joonwon Lee
Computer Architecture Lab. In KAIST

요 약

컴퓨팅 환경이 무선과 휴대용 시스템으로 변화하면서, 전력효율이 점점 중요해지고 있다. 특히 내장형 시스템일 경우에 더욱 그러한데 이중 메모리에서 소모되는 전력이 전체 전력소모의 두 번째 큰 요소가 되고 있다. 메모리 시스템에서의 전력소모를 줄이기 위해서 DRAM의 저전력 모드인 냅모드(nap mode)를 활용할 수 있다. 냅모드는 액티브 모드(active mode)일 때의 28%의 전력만을 소모한다. 하지만 하드웨어 컨트롤러는 운영체제가 협조하지 않으면 이 기능을 효율적으로 활용하지 못한다. 이 논문에서는 DRAM의 액티브 유닛(active unit)의 수를 최소화하는 방법에 초점을 맞춘다. 운영체제는 참조되지 않는 메모리를 냅모드에 놓음으로써 최소한의 유닛들만을 액티브 모드에 놓아 프로그램이 수행될 수 있도록 피지컬(physical) 페이지들을 할당한다. 이것은 PAVM(Power Aware Virtual Memory) 연구의 일반화된 시스템 전반에 대한 연구라고 할 수 있다. 우리는 모든 피지컬 메모리를 고려하고 있으며, 특히 평균적으로 전체 메모리의 절반을 사용하는 버퍼 캐시를 고려하고 있다. 버퍼 캐시의 용량과 그 중요성 때문에 PAVM 방식은 버퍼 캐시를 고려하지 않고는 완전한 해법이 되지 못한다. 이 논문에서 우리는 메모리의 사용처를 분석하고 저전력 페이지 할당 정책을 제안한다. 특히 프로세스의 주소공간에 매핑(mapping)된 페이지들과 버퍼 캐시가 고려된다. 이 두 종류의 페이지들간의 상호작용과 그 관계를 분석하고 저전력을 위해 이러한 관계를 이용한다.

1. 서 론

메모리는 CPU에 이어 에너지를 두 번째로 많이 소모하는 부분이다. 서버환경에서는 메모리는 가장 전력소모가 심한 부분 중 하나이다. 예를 들어 중간 급 IBM eServer 에서는 메모리가 전체 에너지의 40%를 소모한다[1]. 다른 부분들과 달리 메모리는 시스템이 휴지상태일 때도 지속적으로 전력을 소모하므로 메모리에서의 전력소모의 감소는 전체 시스템에 큰 이득이 될 수 있다. 이러한 노력으로 Hai Huang과 그 동료들이 훌륭한 일을 했지만(PAVM)[2,3] 우리는 이 작업이 좀더 시스템 전반에 적용되어야 함을 발견하였다. 특히 버퍼 캐시(buffer cache)가 큰 비중을 차지하므로 버퍼 캐시에 적용하여 큰 효과를 볼 수 있었다. 따라서 이 논문에서는 좀 더 일반화된 방식인 저전력 버퍼 캐시를 제안한다. 각 버퍼들이 속해있는 메모리 영역은 해당 버퍼를 사용할 가능성이 있는 프로세스(process)가 실행될 때에만 액티브 상태에 놓이게 함으로써 최대한 많은 메모리를 저전력 모드에 놓을 수 있게 된다. 이로써 항상 액티브 상태에 있어야 하는 시스템 영역의 크기가 크게 줄어들고 프로세스와 프로세스가 사용하는 버퍼들이 존재하는 메모리 영역만을 최소한으로 액티브 상태에 놓게 된다.

2. 관련 연구

SDRAM 메모리 시스템은 전력제어의 가장 작은 단위인 랭크(rank)들의 집합으로 볼 수 있다. 이러한 랭크는 여러 동작 모드(operating mode)로 동작할 수 있지만[2,3,4,5], 우리는 액티브 모드와 저전력 모드인 냅모드만을 고려하고, 각각을 해당 랭크를 켰다거나 혹은 껐다고 표현한다. 그 외의 동작 모드들은 모드간의 전이에 따른 비용이 크기 때문에 고려하지 않는다. PAVM은 프로세스의 주소공간에 매핑(mapping)된 페이지들을 소수의 랭크에 몰아넣고 이 랭크들만을 액티브 상태에 놓음으로써 전력소모를 크게 줄일 수 있었다. 프로세스의 전환(context switch)때마다 다음 프로세스의 랭크 셋(rank set)만을 켜고 나머지는 꺼놓음으로써 실행시간에 반드시 필요한 랭크들만을 켜놓을 수 있다. 그러나 커널(kernel) 모드에서의 메모리 참조는 고려되지 않았으며 이에 따라 시스템이 사용하는 메모리가 증가할수록 끌 수 없는 시스템 랭크의 수가 증가하게 된다.

그림1은 커널 컴파일(compile)시의 시스템 랭크 셋의 크기이다. 여기서 Normal은 아무런 기법이 쓰이지 않은 수정되지 않은 커널의 경우이다. 이 예에서 볼 수 있듯이 총 8개의 랭크 중 PAVM은 시스템이 사용하는 페이지가 지속적으로 증가하고 있다. 이것은 버퍼 캐시의 양이 점점 증가함에 따른 효과이다. 이러한 현상은 결국 PAVM의 효과를 무효화시키게 된다.

[1] 본 연구는 ITRC 프로그램의 지원을 받아 이루어졌습니다.

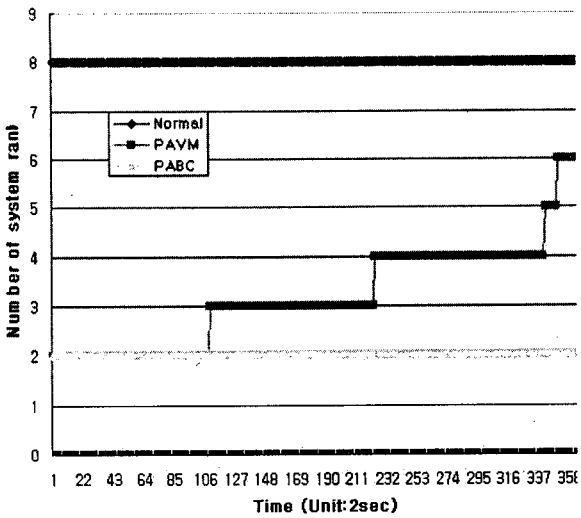

그림1 시간에 따른 시스템 랭크들 크기의 증가

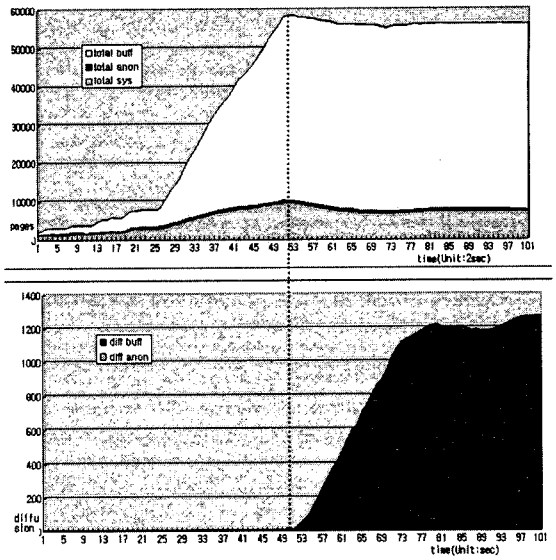

그림2 메모리 부족에 따른 랭크들의 확장

3. 저전력 버퍼 캐시

3.1 기본 디자인

다음과 같이 3가지의 랭크 셋을 정의한다.

α (p) : 프로세스 p의 랭크 셋
β (i) : 아이노드(inode) i에 대한 버퍼 캐시의 랭크 셋
S : 시스템 랭크 셋

α (p) 는 프로세스 p의 주소공간에 매핑된 페이지들이 있는 랭크의 집합이다. β (i) 는 프로세스 p가 열고 있는 파일들에 대한 버퍼들이 있는 랭크의 집합이다. 시스템 랭크는 커널 데이터 구조 등의 시스템이 사용하는 메모리가 있는 랭크들이며 항상 켜져 있게 된다. 이러한 시스템 랭크는 처음 제일 앞의 2개의 랭크로써 정의되며 메모리가 더 필요할 때 확장된다. 이를 바탕으로 프로세스간의 전환할 때마다 다음과 같은 액티브 셋(active set)만을 켜놓고, 다른 랭크들은 꺼놓는다. 우리의 목표는 최종적인 액티브 셋의 크기를 최소화하는 것이고, 이는 다음과 같이 표현된다.

$$\text{Active set size} : | S \cup \alpha(p) \cup \Sigma \beta(i) |$$

이를 위해 커널은 α (p) 와 β (i) 두 종류의 랭크 셋을 유지하며 커널 메모리 할당자는 새로운 페이지가 할당될 때 특정 랭크에서 할당될 수 있도록 수정된다. 이 커널 메모리 할당자에게 α (p) 혹은 β (i)를 힌트로 넘겨줌으로써 각 버퍼와 프로세스의 메모리는 하나의 랭크에 집중될 수 있게 된다. 만약 해당 랭크가 꽉 찼다면 다른 빈 랭크를 찾아서 할당한 뒤 랭크 셋을 확장한다.

3.2 컴팩션(Compaction)

이상적인 형태인 랭크 셋의 크기가 1인 때를 컴팩션이라고 하자. 메모리가 부족해짐에 따라 각 랭크 셋은 크기가 커지기 시작한다. 이러한 현상을 분석하기 위해 확산 값(diffusion)을 다음과 같이 정의한다.

확산 값(Diffusion) = 랭크 셋 크기(rank set size) - 1

이러한 확산 값들의 합을 총 확산 값이라고 할 때, 이 값이 0인 것은 모든 랭크 셋들의 크기가 1임을 알 수 있다. 즉 완전한 컴팩션이 달성된 것이다. 반면 이 값이 클수록 켜놓아야 하는 랭크의 수가 증가한다. 그림2에서 보듯이 버퍼 캐시로 인해 메모리가 부족해지기 시작하자 랭크 셋의 크기가 커지기 시작하고 그 결과 총 확산 값이 증가하고 있다. 이를 해결하기 위해 랭크를 늘리는 시점을 뒤로 미루고 해당 랭크 안의 버퍼 캐시의 페이지 교체(page replacement)를 하게 되면 총 확산 값이 0이 되어 컴팩션을 달성할 수 있다. 이에 따라 다음과 같이 두 가지 정책을 정의한다.

PABC : 랭크 셋의 확장을 항상 허용한다.
PABC-mempol1 : oom killer(out-of-memory killer)를 호출하기 직전에 랭크 셋의 확장을 허용한다.
PABC-mempol2 : 디스크 I/O를 일으키기 직전의 시점에 랭크 셋의 확장을 허용한다.

위와 같이 메모리가 부족할 때 랭크 셋의 확장에 앞서 해당 랭크 내에서 페이지 교체를 먼저 수행한다. 이 두 경우 모두 거의 완전한 컴팩션을 달성할 수 있었다. 실험 결과 이로 인한 버퍼 캐시의 히트 율(hit ratio)의 변화는 미미하거나 오히려 약간 좋아졌다.

3.3 프로세스와 버퍼의 일치

버퍼가 처음으로 할당될 때 해당 버퍼는 버퍼의 할당을 일으킨 프로세스의 랭크들인 α 를 초기값으로 가지도록 한다. 이렇게 함으로써 α (p) 와 해당하는 β (i)를 일치시키게 되고 이것은 액티브 셋의 크기를 최소화한다.

4. 평가

실험은 256MB DDR SDRAM, Pentium 4 2.8GHz CPU, 의 환경에서 수행되었으며, Fedora Core 3에서 Linux 2.6.10의 커널에 구현을 하였다. 하나의 랭크는 32MB로 구성되며, 총 8개의

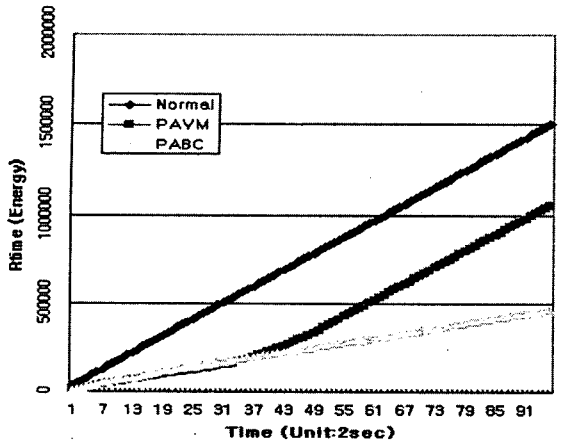

그림3 Diff수행시의 초기 에너지소모량

랭크를 가지고 있다. 측정을 위해 rtime이라는 단위를 다음과 같이 정의한다.

Rtime : 액티브 모드의 하나의 랭크가 1 틱(tick)동안 소모하는 에너지량

리눅스 2.6.10에서 1 틱은 1/1000 초이며, 정의에 따라 rtime은 에너지의 단위가 된다. 각 프로세스는 기존의 utime(user time)과 stime(system time)과 함께 다음과 같은 식으로 rtime을 측정한다.

Rtime += (utime+stime)*active set size

그림3은 두 개의 리눅스 커널 소스에 대해 Diff를 수행하는 경우로서 각 경우의 시간에 대한 rtime을 보여준다. 모든 랭크가 액티브 모드에 놓이는 기존의 커널인 Normal일 때 가장 많은 에너지를 소모하며, PAVM은 부팅 후 약 1분 가량의 시간 동안 에너지 절약을 하지만 곧 기존의 커널일 때와 같은 양상을 보임을 알 수 있다. 반면 PABC에서는 지속적으로 에너지 소모량을 줄이고 있음을 볼 수 있다. 이 그래프는 단일의 디스크 I/O를 많이 하는 프로세스가 수행되는 간단한 상황에서 초기의 PABC의 행동을 잘 보여주고 있다.

그림4와 그림5에서는 실제적인 워크로드(workload)를 사용하는데, 평균적인 사용자의 워크로드를 자동화하여 흉내 내주는 expect 스크립트(script)를 수행한 결과이다. 이 스크립트는 firefox로 인터넷 서핑을 하는 동시에 gimp로 이미지 작업을 하고, pdf문서를 열어 보며 openoffice에서 제공되는 oowriter, oomath, ooimpress, oodraw, oocalc 등을 수행한다. 이러한 작업을 4번 반복하며, 약 20분 정도의 시간을 소모한다.

이러한 실제적인 환경에서 PAVM과 PABC모두 성능이 둔화되어 Normal에 비해 그다지 큰 이득을 얻지 못하고 있다. 이것은 PAVM의 경우 시스템 랭크의 크기가 자라기 때문인 반면, PABC의 경우 랭크들의 확장, 즉 확산 현상 때문이다. 이것을 방지하는 기법을 적용한 PABC-mempol1 과 PABC-mempol2의 경우 다시 한번 크게 전력소모를 줄여주는 것을 볼 수 있다. 이에 따른 히트 율의 변화를 살펴보기 위해 리드어헤드(read ahead)를 끈 상태에서의 히트 율을 측정한 결과 그림5에서와 같이 이로 인한 히트 율의 변화는 거의 없는 것으로 나타난다.

5. 결 론

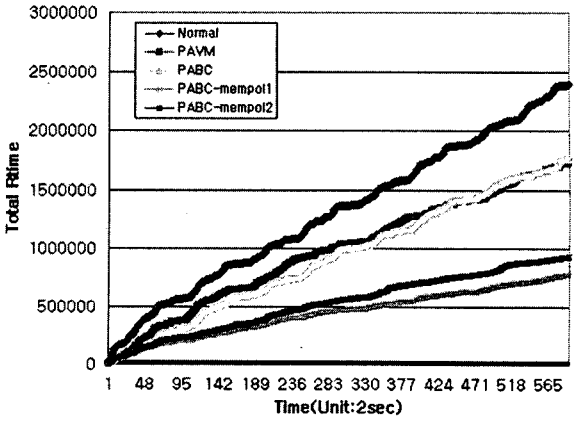

그림4 평균적인 사용자의 워크로드하의 에너지 소모량

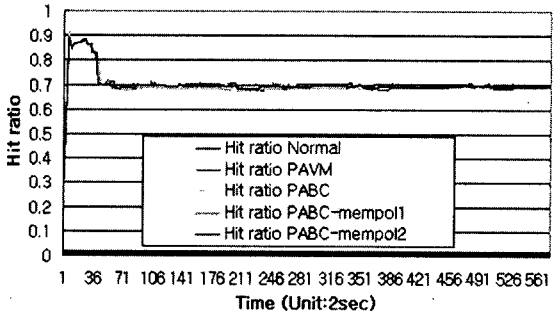

그림5 평균적인 사용자의 워크로드 하에서의 히트 율

우리는 메모리 시스템에서의 전력소모를 줄이기 위해 PAVM의 기법을 일반화하고 특히 버퍼 캐시를 고려하여 PABC를 설계하였다. 시스템 랭크들의 성장을 억제하기 위해서 가장 큰 비중을 차지하는 버퍼 캐시의 할당 위치를 해당 버퍼를 필요로 하는 프로세스가 존재하는 랭크에 위치시켰다. 이것으로 시스템 랭크의 성장을 억제할 수 있었으나 메모리가 부족해지자 컴팩션의 문제가 생겨났고, 이를 해결하기 위해 페이지 교체정책에 약간의 수정을 가하였다. 이를 통해 컴팩션을 달성할 수 있었고 그 결과 평균적인 사용자의 워크로드 하에서 기존의 커널일 때 보다 약 67%가량, PAVM보다는 약 61%가량의 메모리 시스템에서의 에너지 절약을 할 수 있었다.

[1] C. Lefurgy, K. Rajamani, F.Rawson, W. Felter, M. Kistler, and Tom Keller.: Energy management for commercial servers. In IEEE Computer, pages 39-48, Dec 2003.
[2] Hai Huang, Padmanabhan Pillai, Kang G. Shin.: Design and Implementation of Power-Aware Virtual Memory
[3] Hai Huang, Kang G. Shin.:
Cooperative Software-Hardware Power Management for Main Memory.
[4] Delaluz and et al.: Scheduler-based DRAM energy management. In Design Automation Conference 39, 2002.
[5] Alvin R. Lebeck and et al. Power aware page allocation. In Architectural Support for Programming Languages and Operating Systems, pages 105-116, 2000.