리눅스의 실시간 성능 향상을 위한 적응적 커널 쓰레드 제어 기법*

김영곤 김성진O 박찬익
포항공과대학교
younggon@postech.ac.kr, go4real@postech.ac.kr, cipark@postech.ac.kr

An Adaptive Control Method on Kernel Threads to Improve I/O Real-Time Responsiveness in Linux

Younggon Kim Sungjin KimO Chanik Park
Pohang University of Science and Technology

요 약

대부분의 운영체제들은 시스템의 쓰기 성능 향상을 위해서 쓰기 요청에 대해 지연 쓰기 방식으로 처리를 하고 있다. 운영체제의 가상 메모리 관리자는 지연쓰기를 하는 과정에서 발생하는 더티 페이지를, 주기적으로 또는 더티 페이지의 양이 특정 임계치를 초과하였을 때, 이를 담당하는 커널 쓰레드(리눅스의 경우 pdflush)를 통해서 시스템의 전체적인 더티 페이지의 양을 조절하게 된다. 하지만 주기적으로 해당 커널 쓰레드가 실행되지 못하거나, 생성되는 더티 페이지의 급격한 증가로 인해 더티 페이지가 임계치를 초과하게 될 경우 쓰기 요청을 한 태스크에서 직접 더티 페이지를 회수하는 동작을 수행하게 되어 해당 태스크의 쓰기 요청은 지연이 되게 된다. 이는 해당 프로세스의 응답시간에 상당한 증가를 가져오게 된다. 이러한 상황을 개선하기 위해 운영체제의 가상 메모리 관리자는 쓰기 캐시의 현재 상황을 모니터링 하면서 더티 페이지를 디스크에 쓰는 역할을 하는 커널 쓰레드가 적절히 수행되도록 조정할 수 있는 방법이 필요하게 된다. 본 논문에서는 리눅스의 가상 메모리 관리자가 쓰기 캐쉬의 상태를 적절한 상태로 유지할 수 있도록 시스템의 현재 캐쉬의 상태에 기반하여 커널 제어 변수들을 적절히 조절하는 방법을 제안한다. 실험을 통해 본 논문에서 제안한 방법이 시스템의 현재 캐쉬 상태에 따라 변수들을 적응적으로 제어를 하고, 그로 인해 쓰기 응답시간에 상당한 개선을 가져오는 것을 볼 수 있다.

1. 서 론

쓰기 I/O 성능 향상을 위해 대부분의 운영체제는 지연쓰기를 하고 있다. 이는 주 저장장치와 다른 시스템 장치간의 속도 차이에서 발생하는 문제를 개선하여 쓰기성능을 개선할 수 있는 방법으로 여러 번 발생할 수 있는 쓰기 연산을 모아서 처리함으로써 디스크의 접속 횟수를 줄일 수 있게 한 방식이다[4]. 쓰기 과정에서 발생한 더티 페이지들은 가상 메모리 관리자가 디스크에 해당 페이지들을 쓰기 위한 커널 쓰레드를 수행시키기 전까지 쓰기 캐쉬 공간에 머무르게 된다. 오랜 시간 동안 더티 페이지를 쓰기 캐쉬 공간에 두는 것은 쓰기 캐쉬가 일반적으로 비휘발성 메모리를 사용하기 때문에 최근 갱신된 데이터를 손실할 위험성이 증가하게 된다. 반면에 너무 일찍 디스크에 더티 페이지를 쓰게 되면 빈번한 디스크 접근이 이루어지게 되므로 쓰기 캐쉬의 효율을 상당히 떨어뜨리게 된다. 디스크 캐쉬가 읽기와 쓰기 모두를 위해 관리되는 구조에 있어서는 쓰기 캐쉬 공간에 더티 페이지를 많이 유지하는 것이 읽기 캐쉬 공간의 크기를 줄이게 된다. 그러므로 가상 메모리 관리자에게 디스크에 더티 페이지가 쓰여지는 시점을 결정하는 것은 중요한 문제가 된다.

리눅스에서는 가상 메모리 관리자에서 커널 쓰레드(pdflush)가 주기적 또는 설정된 임계치를 초과하였을 경우 더티 페이지를 디스크에 쓴다[4][7]. 디스크에 쓰여지는 더티 페이지의 갯수보다 쓰기 캐쉬에 쓰여지는 더티 페이지의 개수가 더 많을 경우 더티 페이지의 수는 쓰기 공간의 최대 임계값에 도달하게 된다. 이는 (1) 가상 메모리 관리자가 더티 페이지의

본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음" (IITA-2009-C1090-0902-0045)

성장 속도에 맞춰 커널 쓰레드를 깨우는데 실패했거나, (2) 다른 높은 우선 순위의 태스크들로 인해 커널 쓰레드의 실행이 방해가 되는 상황이 발생 하였을 경우 야기 될 수 있다. 더티 페이지가 특정 임계치를 초과하게 되면 write요청을 하던 프로세스에서 더티 페이지를 회수하는 과정이 수행되게 된다. 이 과정은 I/O의 응답시간을 상당히 증가 시키게 된다 (그림 1참조).

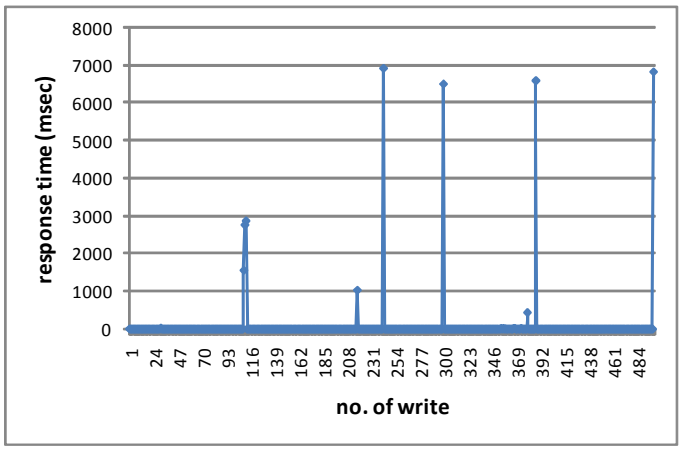


그림 1. 512KB데이터를 쓰기 주기 100ms, 150ms, 200ms로 변경해 가면서 500회 수행한 프로세스의 쓰기 응답시간. Response time이 크게 증가하는 부분은, 프로세스에서 직접 디스크에 더티 페이지를 비우는 과정에서 발생한다.

이는 기존의 쓰기 정책이 더티 페이지의 증가율에 대한 고려가 없고, 또한 pdflush의 우선순위도 정적으로 할당 되어 있기 때문이다. 본 논문에서는 시스템의 상태에 따라 커널 쓰레드(pdflush)의 우선 순위 및 커널 설정 변수들을 동적으로 조정 할 수 있도록 리눅스의 메모리 관리자에 적응적 제어 방법(Adaptive Control Method) 적용을 제안한다. 2장에서는 배경지식과 관련 연구들에 대해 알아보고, 3장에서는 제안한 기법에 대한 내용과 적응적 제어 알고리즘 구현에 대해 알아본다. 그리고 4장에서는 실험을 통하여 제안한 기법을 평가하고 마지막 5장에서 결론을 기술한다.

2. 배경 지식 및 관련 연구

이 장에서는 리눅스에서 기존의 쓰기 캐쉬 비우기 정책(write cache flushing policy) 및 관련 연구에 대해서 설명한다.

pdflush의 동작을 제어하기 위한 네 개의 커널 설정 변수들이 존재한다[4][7].

- vm_dirty_ratio : 더티 페이지로 사용될 수 있는 최대 허용 비율을 정의 한다. 더티 페이지의 비율이 이 임계치를 넘어서게 되면 해당 write요청은 지연되고, 기존 더티 페이지를 일정 기간 동안 디스크에 비우는 동작을 하게 된다.
- dirty_background_ratio : pdflush가 백그라운드로 동작을 하며 더티 페이지를 비우기 시작하는 시점의 비율을 정의 한다.
- dirty_writeback_interval : pdflush가 동작중인 상태에서 각 수행간의 주기를 정의한다.
- dirty_expire_interval : 더티 페이지로 존재할 수 있는 최대 시간을 정의 한다.

그림 2는 네 개의 제어 변수들이 리눅스의 쓰기 캐쉬 비우기 정책에 있어 어떻게 사용되는지를 보여준다. 화살표 파선은 bdi_ratio는 여러 개의 device가 있을 경우 write cache의 공유량을 device별로 할당하기 위해 추가된 변수이다. Device별 할당되는 기준은 모니터링 기간 동안에 disk로 실제 write가 이루어진 값을 통해 정해진다.

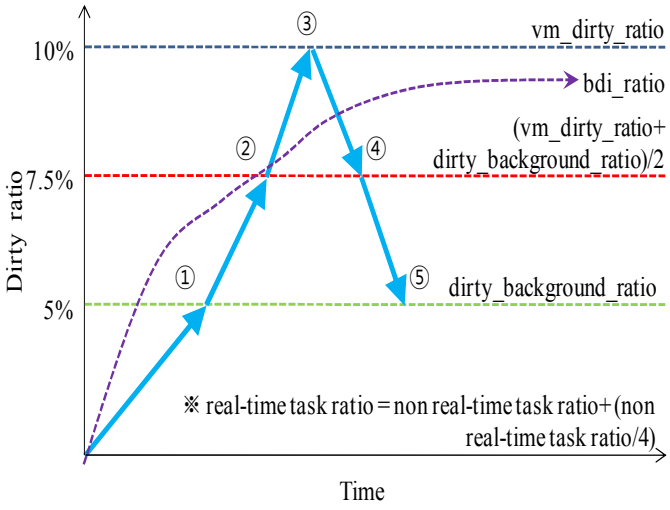


그림 2. 리눅스에서 더티 페이지의 회수 처리 흐름

각 지점의 동작은 다음과 같다.

① pdflush가 동작을 시작한다.
② write요청을 한 프로세스에서 직접 더티 페이지를 디스크에 쓰는 동작을 수행하게 된다.
③ 페이지 캐쉬 내에 더 이상 더티 페이지가 생성되지 않는다.
④ 지연 쓰기가 재게 된다
⑤ 이 지점 이하부터는 pdflush의 동작이 멈추게 된다.

추가적으로 ②, ④지점은 max(bdi_ratio, (vm_drity_ratio + dirty_background_ratio)/2)의 지점을 의미한다. 논문에서는 이 지점을 ***writeout threshold*** 라고 정의한다. ③ 지점은 최대 더티 비율로서 쓰기 캐쉬로 사용될 수 있는 최대치를 의미한다.

플러싱 알고리즘(쓰기 캐쉬로 비휘발성 메모리를 사용하는 경우의 destaging[3])에 관한 이전

연구들에는 DOME[10], high-low watermark algorithm[11], 그리고 AWOL[2] 등이 있다. 그중 [2]와 [9]에 기술된 알고리즘은 유사하며 성능상의 이점을 보였다. 그러나 두 알고리즘 모두 단지 쓰기 I/O를 처리량 향상의 관점에서 커널 제어 변수들을 조정했다. 따라서 pdflush의 정적인 우선순위로 발생할 수 있는 문제점(그림3)은 해결할 수 없다.

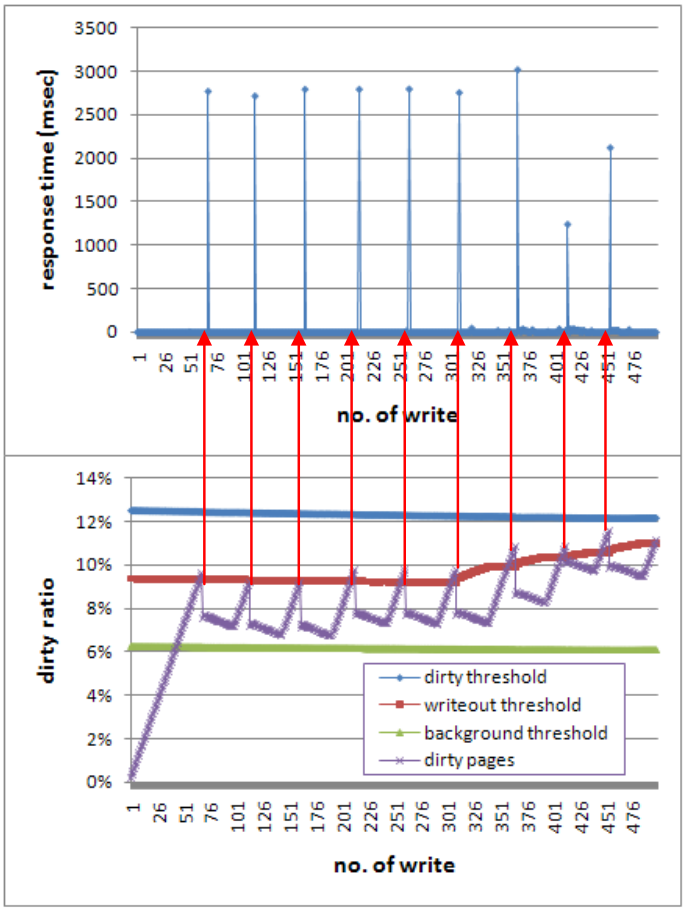


그림 3. 쓰기 I/O응답시간에 pdflush의 정적 우선순위가 미치는 영향. 실험은 실시가 우선순위가 50인 태스크가 512KB 데이터를 100ms, 150ms, 200ms 주기로 변경해 가면서 500회 수행하고, 동시에 실시간 우선 순위가 20인 CPU Bound 태스크를 수행하였다.

그림 3에서 알 수 있듯이, 앞서 정의한 writeout threshold부분까지 pdflush의 정적인 우선순위로 인해 pdflush가 수행이 되지 못하여 더티 페이지 수가 계속 증가함을 볼 수가 있다. writeout threshold 부분에서는 쓰기 요청을 한 태스크에서 직접 실제 디스크로 더티 페이지가 비워지는 동작을 수행하게 되고, 이로 인해 쓰기 요청의 응답시간이 상당히 증가하는 것을 볼 수 있다. 본 논문에서 제안한 알고리즘은 쓰기 I/O의 응답시간이 급격히 증가하는 부분들을 제거하기 위해, pdflush 커널 쓰레드의 우선순위와 쓰기 캐쉬 비우기 정책과 관련된 커널 제어 변수들을 동적으로 변경시키고자 한다.

3. 적응적 제어 기법의 설계 및 구현

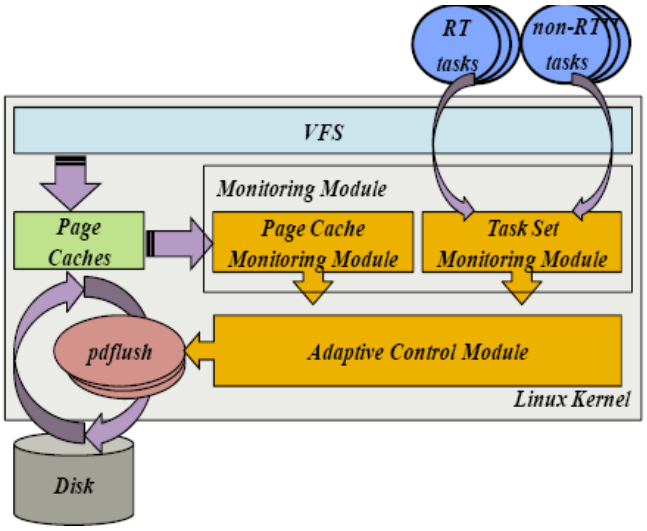


그림 4. 적응적 제어 기법(ACON)의 전체 구조

그림 4는 본 논문에서 제안한 적응적 제어기법의 전체 구조(이후 ACON이라 정의)를 보여준다. Page Cache Monitoring 모듈은 페이지 캐시의 상태를 모니터링하고 Adaptive Control 모듈에 모니터링 된 정보를 제공한다. 유사하게, Task Set Monitoring 모듈도 pdflush 커널 쓰레드의 실행이 요구되는 I/O 요청을 발생한 태스크들의 집합을 수집하고, 수집된 정보를 Adaptive Control 모듈에 전달한다. Adaptive Control 모듈은 Page Cache Monitoring 모듈과 Task Set Monitoring 모듈에 의해 수집된 정보를 기반으로 제어 변수들을 조정한다.

3.1 제어 변수

이전에 언급한 바와 같이, 커널 쓰레드를 깨울 때를 결정하기 위한 제어변수에는 pdflush의 우선순위와 네 개의 커널 변수(vm_dirty_ratio, dirty_background_ratio, dirty_writeback_ratio, dirty_expire_interval)가 있다.

3.1.1 pdflush의 우선순위

pdflush의 우선순위는 pdflush 보다 더 높은 우선순위를 가진 다른 태스크들에 의해 야기되는 스케줄링 지연 문제를 해결하기 위해서 동적으로 조정되는 것이 필요하다[8]. 이를 위해서, 우리는 이전에 개발된 동적 커널 스레드 스케줄링(DKTS) [1] 기법을 수정 적용하였다. 기존 DKTS 기법은 실시간 태스크에 의해 발생된 매 쓰기 요청 마다 우선순위를 조정하지만, ACON에서는 Page Cache Monitoring 모듈로부터 수집된 정보에 기반해 우선순위의 변경이 필요하다고 판단 될 때, 우선순위를 조정한다.

3.1.2 Dirty Ratio

ACON은 더티 페이지의 생성 속도(dirty rate)와 더티 페이지의 플러싱 속도(clean rate) 사이에 비율을 고려하여 vm_dirty_ratio와 dirty_background_ratio의 동적 조정여부를 결정한다. 더티 속도와 클린 속도는 모두 모니터링 모듈에 의해 정보가 수집된다. vm_dirty_ratio를 증가하는 것은 쓰기가 동기적으로 되는 시간을 지연하므로, 그와 비례하게 빠른 쓰기 I/O 응답시간을 유지한다. 그러나 단일 읽기-쓰기 캐시를 사용하는 경우에 읽기 캐시의 크기를 줄이게 된다. 그러므로, vm_dirty_ratio는 페이지 캐시의 40%까지만 증가 할 수 있도록 허용하고, dirty_background_ratio는 (vm_dirty_ratio/2) 까지 설정된다.

3.1.3 Dirty Interval

dirty_expire_interval을 짧게 하는 것은 더티 페이지가 페이지 캐시에 머무는 시간이 짧아지는 것을 의미한다. 그러므로, 더 높은 플러싱 속도가 요구된다면 dirty_expire_interval을 줄이면 된다. 유사하게, dirty_writeback_interval을 줄이는 것은 더 자주 pdflush를 깨어나게 하여 더 높은 플러싱 속도를 얻을 수 있다. 그러나 두 변수의 값을 줄이는 것은 쓰기지연의 효과를 감소시켜 더 높은 디스크 트래픽을 야기할 수 있다.

3.2 Monitoring 모듈

Page Cache Monitoring 모듈은 이전 섹션에서 언급된 제어 변수들에 대한 정보를 측정하고 수집한다. 이를 위해서, 그림 6에서와 같이 커널이 시작하면 페이지 캐시의 모니터링을 시작한다. Page Cache Monitoring 모듈은 다음과 같은 정보를 모니터링 하고 모니터링 한 정보는 미리 지정된 크기의 히스토리를 유지한다.

- 모니터링 주기 당 생성된 더티 페이지의 수
- 모니터링 주기 당 회수된 더티 페이지의 수
- 모니터링 시점에 전체 더티 페이지의 수
- 모니터링 시점에 더티 페이지의 임계치

생성된 더티 페이지의 개수와 회수된 더티 페이지의 개수를 획득하기 위해서, Page Cache Monitoring 모듈은 페이지의 더티 비트(dirty bit)를 설정하고 해제 하는 것을 카운팅 한다. 그리고 전체 더티 페이지의 개수와 더티 페이지 임계치는 기존 커널에서 제공되는 함수를 통해 획득한다. Task Set Monitoring 모듈은 태스크와 pdflush 사이에 관계를 모니터링하고, 만약 pdflush와 관련된 태스크일 경우 태스크 집합에 해당 태스크의 정보를 삽입한다. 이후 이 정보에 기반해서 pdflush의 우선순위를 계산하여 동적으로 할당을 한다.

3.3 Adaptive Control 모듈

Adaptive Control 모듈은 Page Cache Monitoring 모듈과 Task Set Monitoring 모델에서 수집한 정보를 기반으로 제어 플래그를 설정한 후, 설정된 제어 플래그에 따라 각 제어 변수들을 조정한다. Algorithm 1은 모니터링 한 정보를 통해 제어 플래그(cflag)의 비트를 설정하는 알고리즘으로 3.1절에서 살펴본 3개의 변수에 대해 적응적으로 설정을 하고 있다. 모니터링 기간동안 더티 페이지 수와 클린된 페이지 수의 정보에 기반하여 제어 플래그 상태를 결정한다.

```
// cflag는 제어 플래그로 아래 값이 설정
CF_UP_PRIORITY      00000001 /* 우선순위 증가 */
CF_UP_INTERVAL      00000010 /* 쓰기주기 증가 */
CF_UP_THRESHOLD  00000100 /* 더티임계 증가 */
CF_DN_PRIORITY      00001000 /* 우선순위 감소 */
CF_DN_INTERVAL      00010000 /* 쓰기주기 감소 */
CF_DN_THRESHOLD  00100000 /* 더티임계 감소 */
// nr_d와 nr_c는 백그라운드 쓰기주기 동안 더티된 페이지
// 개수와 클린된 페이지 개수
// dt는 더티 페이지 전체 개수
// bg_thresh와 wo_thresh는 각각 백그라운드 임계 비율과
// writeout 임계 비율
// t는 현재 시점, t-1은 바로 이전 시점
 1 if (nr_d == 0 && nr_c == 0)
 2    cflag = CF_DN_PRIORITY | CF_UP_INTERVAL | CF_DN_THRESHOLD
 3 else {
 4    if (nr_d > nr_c && dt(t) > ((bg_thresh+wo_thresh)/2)
 5       cflag = cflag | CF_DN_INTERVAL
 6    else if (nr_d < nr_c && dt(t) > bg_thresh)
 7       cflag = cflag | CF_UP_INTERVAL
 8    if (dt(t) > dt(t-1) && dt(t) > (bg_thresh+wo_thresh)/2)
 9       cflag = cflag | CF_UP_THRESHOLD
10    else if (dt(t) < dt(t-1) && dt(t) > bg_thresh)
11       cflag = cflag | CF_DN_THRESHOLD
12    if (nr_d != 0 && nr_c == 0)
13       cflag = cflag | CF_UP_PRIORITY
14    else if (nr_d == 0 && nr_c != 0)
15       cflag = cflag | CF_DN_PRIORITY
16 }
```

Algorithm 1. 적응적 커널 쓰레드 제어를 위한 제어 플래그 설정 알고리즘

Algorithm 2는 Algorithm 1에 의해 설정된 제어 플래그에 따라 각 제어 변수들을 조정하는 알고리즘이다. pdflush의 우선 순위의 조정과, 최대 최소 크기의 범위 안에서 더티 페이지의 쓰기 기간과 쓰기 만료 기간을 조절하고 있다. 급격히 발생하는 더티 페이지에 효과적으로 대처하기 위해, 쓰기 캐쉬가 증가되는 시점에서는 변수들을 큰 폭으로 증가 시키고 감소되는 시점에서는 적은 폭으로 감소를 시키는 방법을 사용하고 있다.

표 4.1은 각 제어 변수들의 기본값, 최소값, 그리고 최대값을 보여준다.

표 1. 제어 변수의 기본값, 최소값, 그리고 최대값

제어 변수	기본값	최소값	최대값
vm_dirty_ratio	10%	10%	40%
dirty_background_ratio	5%	5%	20%
dirty_writeback_interval	5초	2초	5초
dirty_expire_interval	30초	12초	30초

```
// dwi는 dirty_writeback_interval
// dei는 dirty_expire_interval
// vdr은 vm_dirty_ratio
// bgr은 dirty_background_ratio
// ddr은 vm_dirty_ratio의 초기값
// thresh는 더티 페이지 임계 개수
// d는 vm_dirty_ratio의 초기값에서 임계 개수
// dt는 더티 페이지 전체 개수
// mi는 monitoring interval, 현재 1초
// t는 현재 시점, t-1은 바로 이전 시점
 1 if (cflag & CF_UP_PRIORITY)
 2    dkts_prio_up_all_pdflush();   // dkts 모듈 사용
 3 else if (cflag & CF_DN_PRIORITY)
 4    dkts_prio_down_all_pdflush();
 5 if (cflag == CF_UP_INTERVAL){
 6    dwi(t) = MIN(dwi(t-1)+mi, MAX_DWI);
 7    dei = MIN(dwi(t)*(MAX_DEI/MAX_DWI), MAX_DEI);
 8 }
 9 else if (cflag == CF_DN_INTERVAL){
10    dwi(t) = MAX(dwi(t-1)-mi, MIN_DWI);
11    dei(t) = MAX(dwi(t)*(MAX_DEI/MAX_DWI), MIN_DEI);
12 }
13 if (cflag & CF_UP_THRESHOLD) {
14    vdr(t) = MIN((ddr+(ddr*dt(t)) / thresh(d)), MAX, VDR);
15    bgr(t) = MIN(vdr(t) / (MIN_VDR/MIN_BGR), MAX_BGR);
16 }
17 else if (cflag & CF_DN_THRESHOLD) {
18    vdr(t) = max((vdr(t-1)-(ddr*dt(t)) / thresh(t)), MIN_VDR)
19    bgr = max(vdr(t) / (MIN_VDR/MIN_BGR), MIN_BGR)
20 }
```

Algorithm 2. 적응적 커널 쓰레드 제어를 위한 제어 변수 조정 알고리즘

5. 성능 평가 및 분석

성능 평가를 위해서, 리눅스에 ACON을 구현하였으며, 비교를 위해, 다음과 같은 static kernel과 adaptive kernel에서 동일한 실험을 수행하였다.

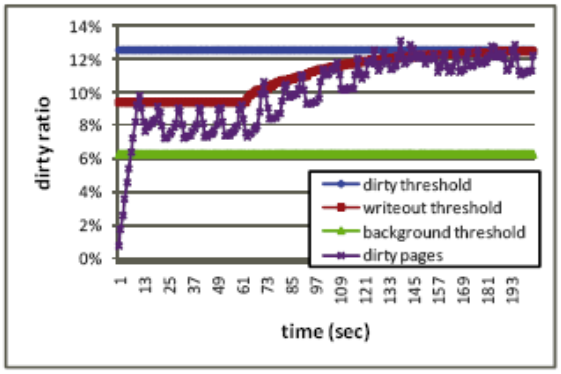
- Static kernel : Vanilla kernel에 real-time patch [5]가 적용된 커널
- Adaptive kernel : Static kernel에 제안한 기법을 적용한 커널

성능 평가를 위해 사용된 장비와 리눅스 커널 그리고 real-time patch의 정보는 다음과 같다.

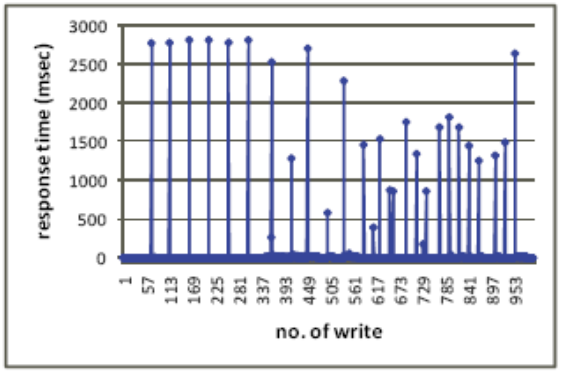
- Intel Pentium IV 2GHz CPU
- 512MB SDRAM
- Ubuntu Linux 6.02
- Linux Kernel 2.6.26
- Real-Time Preemption Patch 2.6.26-rt16

실험에 사용한 상황은 pdflush가 디스크에 더티 페이지를 쓰는 것 보다 빠르게 더티 페이지가 생성되고 동시에 pdflush보다 우선순위보다 높은 CPU Bound작업을 수행하여 pdflush의 동작이 방해되는 상황을 구성 하였다.

실험에 사용한 워크로드는 쓰기 집중적 상황을 위해 실시간 태스크가 512KB 데이터를 100ms, 150ms, 200ms의 주기로 쓰기 요청을 하고, 동시에 pdflush의 스케줄링 지연 상황을 만들기 위해 실시간 우선순위가 20인 CPU-bound 태스크를 같이 동작 시켰다. 그림 5에서 볼 수 있듯이, Static kernel에서는 빠른 더티 속도로 인해 writeout 임계치를 초과한 상황에서 pdflush의 실행마저 지연되어 실시간 태스크가 직접 더티 페이지를 회수하는 상황이 빈번해지고, 그로 인해 많은 부분에서 응답시간이 증가한다. 그러나, 그림 6의 Adaptive kernel에서는 writeout 임계치를 초과하는 상황이 줄어 들고, 그로 인해 실시간 태스크의 응답시간이 현저히 감소한 것을 볼 수 있다. 그림 7 (a)는 Adaptive kernel에서 pdflush의 우선순위 변화를 보여준다. 실행 후 약 10초 후 pdflush의 우선순위가 실시간 태스크의 우선순위(50)를 기반으로 재계산되어 0 에서 26으로 변경된 것을 볼 수 있다. 그리고 그림 7 (b)는 Adaptive kernel에서 dirty_writeback_interval과 dirty_expire_ interval의 변화를 보여준다. 마찬가지로 워크로드에 적응하여 동적으로 변화되는 것을 볼 수 있다.

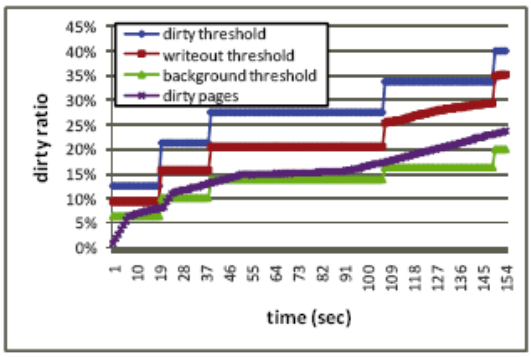


(a) 더티 비율 변화

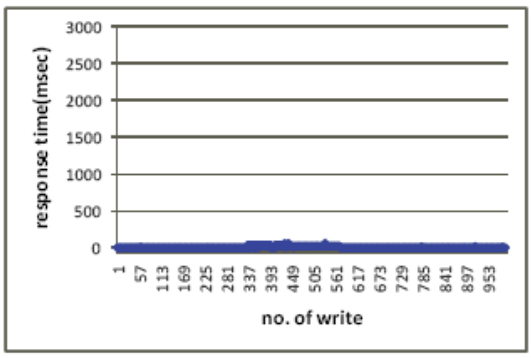


(b) 쓰기 응답시간

그림 5. Static Kernel의 더티 비율 변화 및 쓰기 응답 시간

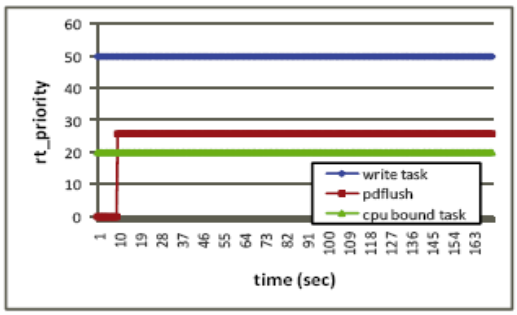


(a) 더티 비율 변화

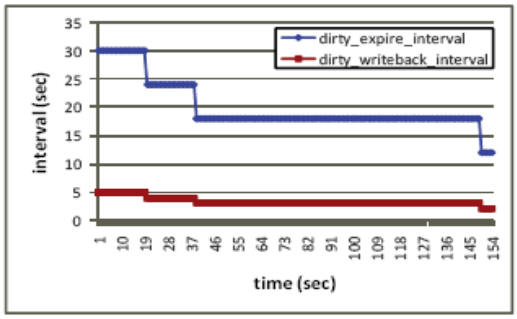


(b) 쓰기 응답시간

그림 6. Adaptive Kernel의 더티 비율 변화 및 쓰기 응답 시간



(a) pdflush의 우선순위 변화



(b) Dirty interval의 변화

그림 7. Adaptive Kernel의 우선 순위 변화 및 dirty interval의 변화

6. 결 론

본 논문에서는 기존 리눅스의 정적인 파일 쓰기 정책의 문제점을 분석 및 제시하였다. 그리고 문제점을 해결하기 위해, 쓰기 캐시의 현재 상태를 고려하여 제어 변수들을 조정하기 위한 적응적 제어 기법을 제안하였다. 제어 변수는 커널 스레드의 실행에 영향을 줄 수 있는 우선순위와 vm_dirty_ratio, dirty_background_ratio, dirty_writeback_interval, dirty_expire_interval을 포함한다.

실험을 통해, 제안한 기법이 문제점을 해결하고 리눅스에서 쓰기 I/O 응답시간을 향상함을 보였다. 그러나 현재 기법은 통합 읽기-쓰기 캐시에 대해서는 고려되지 않아 읽기 I/O 응답시간을 증가할 수 있으므로, 읽기/쓰기 워크로드의 비율과 해당 요청을 발생시킨 프로세스들의 우선순위를 고려하여 파라미터들을 조정 하는 방식으로 이를 개선하는 것이 향후 과제이다.

참고문헌

[1] Dongwook Kang, Woojoong Lee, and Chanik Park, “Kernel Thread Scheduling in Real-Time Linux for Wearable Computers”, ETRI Journal, 2007.

[2] Alexandros Batsakis, Randal Burns, Arkady Kanevsky, James Lentini, and Thomas Talpey, “AWOL: An Adaptive Write Optimization Layer”, 6th USENIX Conference on File and Storage Technologies(FAST), 2008.

[3] A. Varma and Q. Jacobsen, “Destage algorithms for disk arrays with nonvolatile caches”, IEEE Transactions on Computers, 1995.

[4] Bovet and Cesati, “Understanding Linux Kernel 3rd Edition”, O’REILLY, 2006.

[5] Ingo Molnar Real-time Preemption Patch, http://www.kernel.org/pub/linux/kernel/projects/rt

[6] Alessandro Rubini, LINUX DEVICE DRIVERS, O’REILLY, 2000

[7] Norm Murray and Neil Horman, “Understanding Virtual Memory In Red Hat Enterprise Linux 4”, redhat.com, 2005.

[8] L. Sha, R. Rajkumar, and J-P. Lehoczky, ”Priority inheritance protocols: An approach to real-time synchronization”, in IEEE Transactions on Computers, 1990.

[9] Y. J. Nam and C. Park, An adaptive high-low watermark destage algorithm for cached RAID5, In Pacific Rim Internataional Symposium on Dependable Computing, 2002

[10] M. Alonso, V. Santonja and C. de Vera, A new destage algorithm for disk cache: DOME, PEUROMICRO Conference, 1999

[11] J. Menon and J. Cortney, The architecture of a faulttolerant cached raid controller, In Proceedings of the 20th International Symposium on Computer Architecture, 1993