리눅스 운영체제하의 OLTP환경에서 RAID 레벨 5의 효율적인 캐쉬 운영 방안에 대한 연구

송자영° 장태무
동국대학교 컴퓨터공학과
{songjy, jtm}@dgu.ac.kr

A study on Efficient Management of RAID Level 5 Cache in OLTP Environment under Linux Operating System

Jayoung Song° Taemu Chang
Dept. of Computer Engineering, Dongguk University

요 약

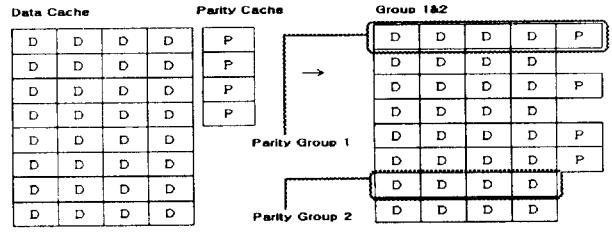
RAID 레벨 5는 쓰기 시에 패리티 갱신을 위한 4번의 디스크 접근으로 인하여, OLTP와 같이 상대적으로 빈번한 디스크 접근을 가지고 데이터 크기가 작으며 쓰기의 횟수가 많은 작업환경에서 성능이 떨어지게 된다. 데이터와 패리티에 대한 캐싱은 OLTP환경에서의 쓰기에 대한 문제를 해결하기 위한 기법이다. 본 논문에서는 리눅스 운영체제의 파일 데이터 구조에 변화를 주고, 커널에서 얻어진 정보를 디스크 캐쉬의 운영에 이용한다. 스트라입 크기(G)를 가지는 RAID 레벨 5에서 패리티 캐쉬의 크기가 전체 캐쉬 크기의 1/G 이하 일 경우 데이터 패리티 캐쉬 크기 변화에 영향을 받지 않고 캐쉬의 그룹 단위 운영과 그에 따른 패리티의 미리 읽기를 가능하게 하여 패리티에 대한 추가적인 읽기를 최소화하는 기법을 제안한다. 본 논문의 실험 결과는 초당 디스크에 도착하는 평균 디스크 접근 요구 개수에 변화를 주어 시뮬레이션 방법으로 입증하였으며, OLTP 환경에서 데이터와 패리티 캐쉬를 독립적으로 운영하는 일반적인 캐쉬 운영 방법에 비해 평균 응답시간을 단축시킬 수 있음을 알 수 있다.

1. 서론

1980년도 후반 이후, 매년 20~30%씩 증가되는 프로세서의 처리 속도에 비하여 디스크의 접근 속도의 향상은 5% 정도에 불과하다[1]. 이로 인해 발생되는 입출력 위기를 극복하기 위해 현재 가장 널리 사용되고 있는 저장수단이 RAID(Redundant Arrays of Inexpensive Disks)이다[2]. RAID는 일반적으로 신뢰도를 높이기 위한 추가정보의 배치 방법에 따라 여러 단계로 나누어지며, 일반적으로 1~6단계가 많이 이용되고 있다[3]. 물류관리와 같이 한번에 요구되는 데이터 크기가 작고 입출력 횟수가 많은 OLTP(On-Line Transaction Processing)작업 환경에서는 레벨 5가 적합한 특성을 가지고 있다[3]. 데이터와 보조 정보인 패리티를 분산시켜 저장하는 레벨 5는 신뢰도와 성능 면에서 OLTP 환경에 적합한 구조를 가지고 있으나, 쓰기 시에 패리티 갱신을 위한 4번의 디스크 접근으로 인하여 성능 면에서 오버헤드가 되고 있다. 레벨 5에서 쓰기 시, 성능의 오버헤드를 극복하기 위하여 Parity caching, Parity logging, Floating parity등의 기법 등이 제안되었다[4][5].

본 논문에서는 리눅스 운영체제하의 OLTP 환경에서 디스크의 응답시간을 높일 수 있는 캐쉬 운영 방법인 APGOC(Adaptive Parity Group On Cache)를 제안하고 시뮬레이션을 통하여 일반적인 캐쉬 운영방법과 비교함으로써 그 성능을 검증한다. APGOC는 디스크 제어기에 존재하는 디스크 캐쉬의 새로운 운영 방법이다. 리눅스와 유닉스에서 사용자 프로세스가 한 파일에 대한 접근을 커널에 신청할 경우 파일 데이터 구조를 통하여 그 파일에 대한 접근을 하게 된다[6][7]. APGOC는 리눅스 파일 시스템의 파일 데이터 구조에 디스크 캐쉬 운영에 관한 정보를 추가하고, 디바이스 드라이버를 통하여 디스크 제어기에 디스크 접근 요청을 할 때 그 정보를 함께 보내어 캐쉬의 운영에 이용하는 디스크 캐쉬 운영 방법이다.

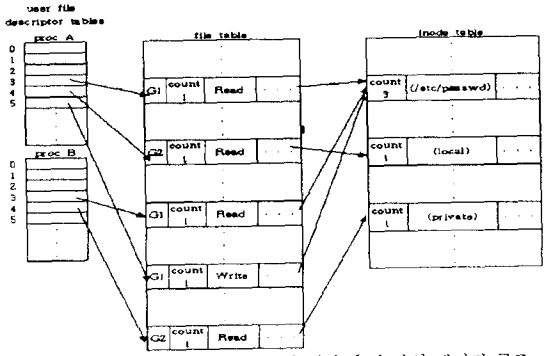
2. APGOC(Adaptive Parity Group On Cache)



[그림 1] APGOC의 논리적 캐쉬 라인

APGOC는 캐쉬 운영 단위를 [그림 1]과 같이 Group1과 Group2로 분리하여 운영한다. Group1은 패리티 블록을 포함

하고 있는 캐쉬 라인이고 Group2는 데이터 블록들만 포함하고 있는 캐쉬 라인이다. 커널은 사용자 프로세스의 파일접근에 대한 요청이 들어오면 [그림 2]와 같이 Group1과 Group2 캐쉬 라인 운영을 위한 정보를 포함한 파일 데이터 구조를 생성한다. 데이터 구조 생성 시, 커널에서는 사용자 프로세스의 접근 권한과 Inode에 존재하는 파일의 접근 권한을 비교 한다. 일치하는 Inode의 권한에 쓰기 속성이 있을 경우 Group1으로 설정하고 없을 경우는 Group2로 설정한다. [그림 3]의 알고리즘이 리눅스의 파일 열기 알고리즘에 삽입되어 Group1과 Group2의 설정을 담당하게 된다. 그 후 사용자가 접근하려는 블록의 Group1과 Group2에 대한 정보를 수정된 파일 데이터 구조, [그림 2]의 파일 테이블에서 찾아 디스크 접근 시에 함께 디스크 제어기에 넘겨주게 된다.



[그림 2] 두 프로세스에 의한 파일 개방 후의 파일 데이터 구조

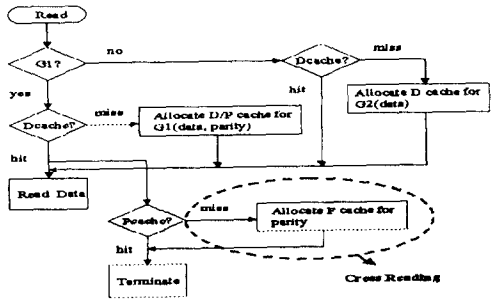
```
Algorithm initindex
 Input :   α . Working inode
           β . Working file table
 ouput :   none
 {
      if(user id of current process == user id of inode)        /* User */
           if(user permission of inode have W-permission)
                disk request index of file table = Group1;
           else
                disk request index of file table = Group2;
      else if(group id of current process == group id of inode) /* Group */
           if(group permission of inode have W-permission)
                disk request index of file table = Group1;
           else
                disk request index of file table = Group2;
      else                                                      /* Other */
           if(other permission of inode have W-permission)
                disk request index of file table = Group1;
           else
                disk request index of file table = Group2;
 }
```

[그림 3] Group1 과 Group2 설정 알고리즘

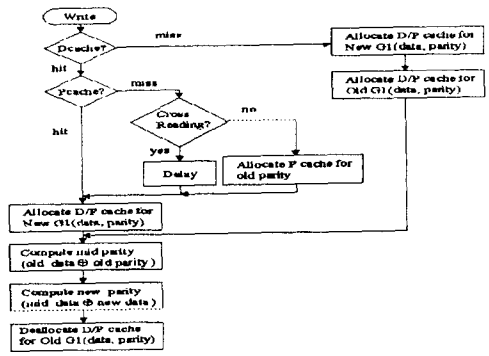
디스크 데이터의 읽기 요구가 발생했을 때 APGOC의 캐쉬 동작은 [그림 4]와 같다. 읽기의 경우 Group1 읽기와 Group2 읽기로 나누어져 있다. 읽기에서 중요한 부분은 Cross 읽기 이다. Cross는 Group1 읽기의 경우에, Group2 읽기로 읽어온 블록이 캐쉬 상에 존재하고 그것과 일치하는 패리티 블록이 패리티 캐쉬 상에 존재하지 않을 경우 발생한다. 결국 이 경우는 패리티의 추가적인 읽기가 발생 할 수 있다. 아래의 경우는 리눅스에서 Cross가 발생할 수 있는 예를 보여준다.

```
-rwxr--r--  1 songjy staff simul.c
```

Other인 권한을 가진 사용자가 파일을 읽었을 경우 캐쉬에는 Group2 읽기로 인해 데이터만 가지고 온다. 그 후 songjy가 동일한 파일에 대해 쓰기를 시도할 경우 Group1 접근으로 인해 패리티 블록을 패리티 캐쉬에 적재 해야 한다. 디스크 데이터의 쓰기 요구가 발생했을 때 APGOC의 캐쉬 동작은 [그림 5]와 같다. APGOC는 패리티 연산을 위해 이전 데이터와 새로운 데이터 모두 캐쉬에 저장한다. 즉, RAID 레벨 5의 읽기-수정-쓰기(read-modify-write) 방식을 사용하여 동작하며 패리티 연산은 패리티 엔진에서 하드웨어적으로 이루어진다.



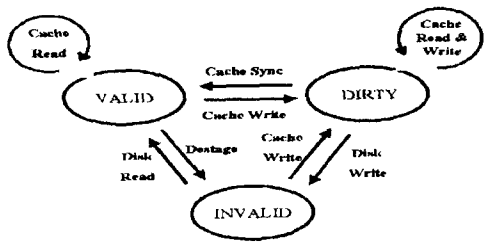
[그림 4] 디스크 데이터의 읽기 흐름도



[그림 5] 디스크 데이터의 쓰기 흐름도

APGOC를 위한 데이터 캐쉬는 기존의 캐쉬 상태정보에 추가하여 Group1과 Group2를 위한 상태정보를 캐쉬라인에 가지고 있어야 하며 패리티 캐쉬는 Cross 읽기를 위한 상태정보를 캐쉬라인에 가지고 있어야 한다.

[그림 6]은 APGOC에서 Group1과 Group2에 모두 동일하게 적용되는 캐쉬 라인의 상태 천이를 나타낸다.



[그림 6] APGOC의 캐쉬라인 상태 천이도

캐쉬 라인의 각 상태는 다음과 같이 정의한다.

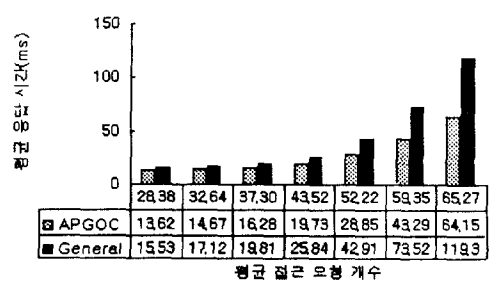
- VALID : 라인을 구성하는 모든 블록이 유효할 때
- DIRTY : 라인을 구성하는 블록 중에 하나라도 dirty 상태인 경우
- INVALID: 라인을 구성하는 모든 블록이 유효하지 않을 때

3. 성능 분석

본 논문에서는 2장에서 제안한 APGOC가 기존의 데이터와 패리티 캐쉬를 독립적으로 운영하는 일반적인 캐쉬 운영방법 보다 성능이 개선됨을 보이기 위하여 시뮬레이션 방법을 사용하였다. 사용한 시뮬레이터는 작업 부하로 DiskSim 1.0의 기본 트레이스 데이터[8]를 OLTP 환경에 맞도록 수정하여 사용하였으며, C 언어로 시뮬레이션 모델을 구현하여 응답시간을 측정하였다. 본 논문에서 사용한 시뮬레이션 변수는 [표 1]과 같으며 패리티 캐쉬의 용량은 전체 캐쉬 용량의 10분의 1로 하였다. 작업 부하에서 Group1과 Group2는 파일 접근의 시간적 국부성을 고려하여 다음의 알고리즘에 의해 추가하였다.

[표 1] 시뮬레이션 변수

Disk Parameter	
Disk Name:	IBM Lightning 0661 311MB drive
Geometry:	949cyls, 14 heads, 48 sectors/track
Sector Size:	512bytes
Revolution Time:	13.9ms
Seek Time Model:	2.0 + 0.01 · dist + 0.46 · (dist)$^{0.5}$ dist is cylinder number traversed 2ms min, 12.5ms avg, 25ms max
Array Parameter	
Head Scheduling :	FCFS
Disk Number:	5
Workload Parameter	
Req/sec per disk(avg):	28.38, 32.64 , 37.30, 43.52, 52.22, 59.35, 65.27
Request Size:	Fixed 2KB
R/W Ratio:	4:1
Request Number:	100000
Cache Parameter	
Stripe Unit Size:	512bytes
Cache Size:	0.2%
Scheduling Algorithm:	LRU
Group1 Size:	2.5KB
Group2(data) Size:	2KB



[그림 7] 시뮬레이션 결과

작업 부하 발생기는 Disksim 1.0[8]의 데이터를 분석하여 쓰기 요청을 발견하면, 그 쓰기 요청 시간으로부터 일정한 상한 하한 시간 사이를 검사하고, 쓰기 요청 블록 번호들과 같은 블록 번호를 가지는 접근 요청이 검색되는 경우는 Group1으로 그 외의 경우는 Group2로 지정한다. 시뮬레이션 결과는 [그림 7]과 같다. [그림 7]은 디스크 마다 요구되는 1초당 평균 접근 요청 개수의 변화에 따른 평균 응답시간을 나타낸 것이다. APGOC와 일반적인 캐쉬의 운영 방법을 비교하여 볼 경우, [표 1]의 환경에서 평균 요청의 개수가 높아질수록 평균 응답 시간 면에서 높은 성능 향상을 보임을 알 수 있다.

4. 결론

APGOC는 스트라입 크기(G)를 가지는 RAID 레벨 5에서 패리티 캐쉬의 크기가 전체 캐쉬 크기의 1/G 이하 일 경우 데이터 패리티 캐쉬 크기 변화에 영향을 받지 않고 캐쉬의 그룹 단위 운영을 가능하게 한다. 특정 사용자가 접근하려는 파일에 대한 쓰기 가능 여부를 미리 파악하고, 그것에 대한 정보를 캐쉬가 존재하는 디스크 제어기에 알려줌으로써, 쓰기 가능 파일에 관련된 블록을 캐쉬로 읽어올 경우 패리티마저도 읽어온다. 결국 동일한 블록에 대하여 쓰기가 발생할 경우에도 패리티 미리 읽기로 인하여, 추가적인 패리티의 읽기로 인한 지연을 방지할 수 있고 그룹단위 운영으로 인해 데이터와 패리티의 연동 운영을 통한 RAID 레벨 5의 병렬성을 최대로 활용하는 것이 가능하게 된다. OLTP와 같은 짧은 시간 동안 많은 디스크 접근을 필요로 하는 환경에서 APGOC는 기존의 데이터 패리티 캐쉬를 독립적으로 운영하는 일반적인 캐쉬 운영 방법에 비하여 높은 효율을 기대할 수 있다.

참고 문헌

[1] M. C. Holland, On-Line Data Reconstruction in Redundant Disk Arrays, PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1994.

[2] G. A. Gibson, Tutorial on Storage Technology: RAID and Beyond, Proceeding of the ACM SIGMOD International Conference on Management of Data, 1995.

[3] D. Patterson et al, A case for Redundant Arrays of Inexpensive Disks, Proceeding of the ACM SIGMOD International Conference on Management of Data, pp.109-116, 1988.

[4] Daniel Stodolsky, Parity Logging Overcoming the Small Write Problem in Redundant Disk Arrays, International Symposium on Computer Architecture, 1993.

[5] Sunil K. Mishra and Prasant Mohapatra, Performance Study of RAID-5 Disk Arrays with Data and Parity Cache, International Conference on Parallel Processing, pp.222-209, 1996.

[6] M. J. Bach, The Design of The Unix Operating system 2th, Prentice Hall International, 1986.

[7] Beck Michael, Linux Kernel Internals, Addison-Wesley Pub, 1997.

[8] G. R. Ganger et al, The DiskSim Simulation Environment Version 1.0 Reference Manual, Technical Report CSE-TR-358-98, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1998.