論文 97-22-7-24

소프트웨어 RAID 파일 시스템에서 오손 블록 교체시에 효율적인 캐싱 기법

正會員 김 종 훈*, 노 삼 혁*, 원 유 헌*

An Efficient Caching Scheme at Replacing a Dirty Block for Software RAID File Systems

Jong-Hoon Kim*, Sam H. Noh*, Yoo-Hun Won** *Regular Members*

※본 논문은 1996년 한국학술진흥재단의 공모 과제 연구비에 의해 연구되었음.

요 약

소프트웨어 RAID 파일 시스템이란 네트워크로 연결된 다수의 시스템들이 보유한 각 디스크를 통해 하드웨어 RAID의 기능을 소프트웨어적으로 제공하는 시스템을 의미한다. 이러한 시스템은 기존의 파일 시스템에 비해 높은 성능과 신뢰성을 제공해준다. 본 논문에서는 소프트웨어 RAID 파일 시스템에서 효율적인 캐싱 기법을 제안한다. 그리고 이와 기존의 캐싱 기법들을 소프트웨어 RAID 파일 시스템에 적용한 정책들과의 성능을 비교한다. 성능 분석을 통해 우선 소프트웨어 RAID 파일 시스템에서 작은 쓰기 동작은 성능을 크게 저하시키는 요소임을 확인할 수 있었다. 이러한 문제를 해결하기 위해 캐쉬 영역을 논리적인 두 단계로 구성하여 관리하는 캐싱 기법을 제안하였다. 그중 한 영역에 옛 데이터와 패리티 정보를 유지시킴으로 오손(dirty) 블록 교체시에 발생하는 네 번의 디스크에 대한 요구를 최소화시켜 줌으로 성능을 향상시키게 된다. 이러한 캐싱 기법들에 대한 성능 비교는 트레이스 기반 시뮬레이션에 의해 수행되었다. 비교 결과를 통해 본 논문에서 제안한 캐싱 기법이 기존의 정책들에 비해 효율적인 성능을 나타냄을 확인할 수 있었다.

ABSTRACT

The software RAID file system is defined as the system which distributes data redundantly across an array of disks attached to each workstations connected on a high-speed network. This provides high throughput as well as higher availability. In this paper, We present an efficient caching scheme for the software RAID file system. The performance of this scheme is compared to two other schemes previously proposed for conventional file systems and adapted for the software RAID file system. As in hardware RAID systems, small-writes to be the performance

*홍익대학교 컴퓨터공학과

論文番號:97188-0603
接受日字:1997年 6月 3日

bottleneck in software RAID file systems. To tackle this problem, we logically divide the cache into two levels. By keeping old data and parity values in the second-level cache we were able to eliminate much of the extra disk reads and writes necessary for write-back of dirty blocks. Using trace driven simulations we show that the proposed scheme improves performance for both the average response time and the average system busy time.

Ⅰ. 서 론

NFS[1], Sprite[2]와 같은 전통적인 클라이언트/서버 파일 시스템에서는 모든 파일 시스템에 대한 서비스를 중앙 서버가 제공해 준다. 그러므로 이러한 클라이언트/서버 형태의 분산 파일 시스템에서는 서버에 병목현상이 발생한다. 반면 Zebra[3]와 xFS[4]와 같은 소프트웨어 RAID 파일 시스템[3, 4, 5, 6]은 파일 저장시 여러 워크스테이션의 디스크에 분산시켜 동시에 저장함으로 성능 향상을 가져옴과 동시에 특정 기계에 대한 부하를 줄이며, 또한 패리티 정보를 추가적으로 저장함으로 시스템에 결함이 발생하여도 신뢰성 있게 동작하는 분산 파일 시스템이다. 소프트웨어 RAID 파일 시스템에 관한 상세한 내용은 [6]을 참조하기 바란다.

이러한 소프트웨어 RAID 파일 시스템의 성능을 최적화하기 위해서는 캐쉬의 사용이 불가피하다. 그러나 소프트웨어 RAID 파일 시스템은 기존의 분산 파일 시스템에서의 캐싱 동작과는 큰 차이가 있는데 이는 오손(dirty) 블록의 교체가 발생할 경우이다. 본 논문에서는 소프트웨어 RAID 파일 시스템에서 오손 블록 교체시에 효율적으로 동작하는 캐싱 기법을 제안하고 기존의 캐싱 기법들을 소프트웨어 RAID 파일 시스템에 적용한 기법들과의 성능 분석을 하여 성능을 비교한다. 이를 위하여 소프트웨어 RAID 파일 시스템을 세부적으로 모델링한 시뮬레이터를 개발하였다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 소프트웨어 RAID 파일 시스템에 캐쉬를 적용하였을 때의 변화와 본 논문에서 제안하는 캐싱 기법과 기존의 캐싱 기법을 소프트웨어 RAID 파일 시스템에 적용한 기법들에 대해 설명한다. Ⅲ장에서는 성능 평가를 위한 실험 환경을 기술하며 실험 결과와 분석 내용을 설명한다. 그리고 Ⅳ장에서 결론을 맺는다.

Ⅱ. 소프트웨어 RAID 파일 시스템에서 캐싱

본 장에서는 소프트웨어 RAID 파일 시스템 환경에 캐쉬를 적용하였을 때 어떠한 변화가 생기는지를 살펴보고 본 논문에서 제안한 캐싱 기법과 기존의 캐싱 기법을 소프트웨어 RAID 파일 시스템에 적용한 기법들을 살펴본다. 우선 본 논문의 시스템 환경은 [6]을 기반으로 하고 있으므로 [6]을 참조하기 바란다.

2.1 캐 싱

파일 시스템에서 성능을 최적화 하기 위해서는 캐쉬의 사용이 불가피하다. 그러나 소프트웨어 RAID 파일 시스템에 캐쉬를 적용하게되면 기존 시스템과는 다르게 동작을 하게 되는데 이는 각 워크스테이션 캐쉬에서 오손 블록 교체시의 동작이다. 그 동작을 살펴보면 다음과 같다.

(1) 교체 블록의 옛 데이터와 교체 블록과 같은 패리티 그룹[7]에 속하는 옛 패리티 읽기
(2) 새로운 데이터(교체 블록)와 옛 데이터, 그리고 옛 패리티를 가지고 새로운 패리티 계산
(3) 새로운 데이터와 새로운 패리티 쓰기

이러한 동작을 작은 쓰기 동작[7]이라 하고 하나의 작은 쓰기 동작을 처리하기 위해서는 옛 데이터 읽기, 옛 패리티 읽기, 새로운 데이터 쓰기, 새로운 패리티 쓰기의 네 번의 디스크 요구가 수반된다. 하드웨어 RAID에서 이러한 동작은 응답 시간(response time)을 RAID 레벨 0에 비해 약 두 배 가량 증가시키고 처리량(throughput)에 있어서는 약 1/4 가량으로 감소시킨다[7]. 많은 작은 쓰기 동작을 발생시키는 트랜잭션 처리와 같은 응용에 있어서 RAID 레벨 4와 5의 이와 같은 성능의 감소는 RAID 레벨 0과 1에 비해 너무도 큰 것이다. 이러한 작은 쓰기 동작에서의 성능을 증가시키기 위해 기존의 하드웨어 RAID에서는

Parity Logging[8], Floating Parity[9], 그리고 비휘발성 메모리를 이용하는 방법[10]등과 같은 연구가 진행되었다. 그러나 이러한 하드웨어 RAID에서의 기법을 소프트웨어 RAID에 그대로 적용할 수 없으므로 본 논문에서는 소프트웨어 RAID 파일 시스템 환경에 효율적인 캐싱 기법을 제안한다.

2.2 캐싱 기법

본 절에서는 우선 기존 분산 파일 시스템에서 적용하던 캐싱 기법을 소프트웨어 RAID 파일 시스템에 적용한 두가지 기법과 본 논문에서 제안하는 오손 블록 교체시에 효율적인 캐싱 기법에 대하여 살펴본다. 모든 경우의 캐쉬 교체 정책은 LRU라 가정하였다.

2.2.1 다중 복사 캐싱 기법

다중 복사 캐싱 기법은 같은 블록을 여러 워크스테이션에서 중복해서 저장이 가능하게 하도록 하는 방법으로 리모트 접근을 줄이고자 하는 기법이다[1]. 동일한 블록이 여러 개의 캐쉬에 저장됨으로 인해 발생하는 캐쉬 일관성에 대한 문제는 쓰기-무효화(write-invalidation) 정책을 이용하여 해결한다.

블록 요구시 다음의 단계로 동작한다.

```
if (로칼 캐쉬에서 적중(hit)) then
   LRU 리스트에 MRU 블록으로 위치 ;
else if (리모트 캐쉬에서 적중) then {
   리모트 캐쉬의 LRU 리스트는 원래의 상태 유지 ;
   복사해서 로칼 캐쉬의 MRU 블록으로 위치 ;
}
else if (로칼 디스크에 존재) then
   복사해서 로칼 캐쉬의 MRU 블록으로 위치 ;
else if (리모트 디스크에 존재) then {
   리모트 캐쉬의 MRU 블록으로 위치 ;
   복사해서 로칼 캐쉬의 MRU 블록으로 위치;
}
```

교체될 블록이 오손인 경우 새로운 패리티를 계산하기 위하여 옛 데이터와 옛 패리티를 읽어오고 새로운 블록과 새롭게 계산된 패리티를 써야 하는데 본 기법에서는 이러한 모든 입출력 동작이 디스크에서 발생한다.

2.2.2 단일 복사 캐싱 기법

캐쉬 자체의 구성 형태는 다중 복사 캐싱 기법과 같으나 시스템 전체의 캐쉬 내에 저장되어 있는 모든 블록은 한 개만을 유지시키는 기법이다[11]. 본 기법이 제안된 동기는 다음과 같다. 다중 복사 캐싱 기법에서는 동일한 블록이 중복해서 저장됨으로 그런 만큼의 다른 유용한 블록을 캐쉬에 두지 못하는 문제점이 있는데 이를 해결하기 위함이며, 최근 초고속의 네트워크의 출현으로 리모트 캐쉬로부터 로칼 캐쉬로 데이터를 이동시키는 것은 성능에 큰 영향을 미치지 않는다는 이유 때문이다. 또한 다중 복사 캐싱 기법에서 대두되던 캐쉬 일관성에 대한 문제는 발생하지 않게 된다.

블록 요구시 다음의 단계로 동작한다.

```
if (로칼 캐쉬에서 적중(hit)) then
   LRU 리스트에 MRU 블록으로 위치 ;
else if (리모트 캐쉬에서 적중) then {
   리모트 캐쉬의 LRU 리스트에서 제거 ;
   이동시켜 로칼 캐쉬의 MRU 블록으로 위치 ;
}
else (* 디스크 *)
   복사해서 로칼 캐쉬의 MRU 블록으로 위치 ;
```

다중 복사 캐싱 기법과 동일하게 오손 블록이 교체될 때 네 번의 입출력 요구가 모두 디스크에서 발생한다.

2.2.3 2단계 캐싱 기법

앞서 언급했듯이 전체 시스템의 병목으로 디스크의 입출력이 지목되고 있는 가운데 교체될 블록이 오손인 경우 네 번의 입출력 동작이 발생하게 되는데 이러한 동작이 모두 디스크에서 발생하게 되면 전체 시스템의 성능이 저하됨은 자명한 사실이다. 앞에서 살펴본 두 가지 기법은 네 번의 입출력이 모두 디스크로의 접근을 유발시키는 기법이다. 본 기법은 오손 블록이 교체될 경우 디스크에 대한 접근을 최소화하여 성능을 향상시키고자 제안된 기법으로 캐쉬를 논리적인 두 단계로 구성하고 각 단계의 캐쉬마다 역할을 달리한다. 첫 번째 단계의 캐쉬는 기존의 캐싱 기법과 같은 동작을 취하나 두 번째 단계의 캐쉬는 쓰기 요구

가 발생하여 데이터를 디스크로부터 캐쉬로 복사해 올 경우 옛 데이터를 저장하고 있으며, 패리티 블록도 저장하고 있음으로 작은 쓰기 동작이 발생하였을 때 디스크로의 접근을 최소화시켜 주는 영역이다.

블록 요구시의 동작을 살펴보면 다음과 같다.

if (로칼 1-단계 캐쉬에서 적중) then
LRU 리스트에 MRU 블록으로 위치;
else if (리모트 1-단계 캐쉬에서 적중) then {
리모트 1-단계 캐쉬의 LRU 리스트에서 제거;
이동시켜 로칼 1-단계 캐쉬의 MRU 블록으로 위치;
}
else if (로칼 혹은 리모트 2-단계 캐쉬에서 적중) then
복사해서 로칼 1-단계 캐쉬의 MRU 블록으로 위치;
*else (*리모트 또는 로칼 디스크에 존재 *) {*
if (읽기 요구) then
복사해서 로칼 1-단계 캐쉬의 MRU 블록으로 위치;
*else (*쓰기 요구 *) {*
복사해서 2-단계 캐쉬의 MRU 블록으로 위치;
복사해서 로칼 1-단계 캐쉬의 MRU 블록으로 위치;
쓰기 동작은 1-단계 캐쉬 내의 블록에서만 발생;
}
}

1-단계 캐쉬들간의 동작은 단일 복사 캐싱 기법과 같으나, 2-단계 캐쉬는 자기 자신의 디스크에 저장되어 있는 블록만을 저장하며 2-단계 캐쉬들간 블록 이동은 발생하지 않는다.

1-단계 캐쉬에서 오손 블록이 교체될 때의 동작을 살펴보면 다음과 같다.

if (옛 블록이 워크스테이션의 2-단계 캐쉬에 존재) then
2-단계 캐쉬로부터 복사해서 로칼 워크스테이션으로 읽어 옴;
else
해당 블록이 저장되어 있는 워크스테이션의 디스크로부터 복사해서 로칼 워크스테이션으로 읽어 옴;
if (패리티 블록이 워크스테이션의 2-단계 캐쉬에 존재) then
2-단계 캐쉬로부터 복사해서 로칼 워크스테이션으로 읽어 옴;
else
해당 패리티 블록이 저장되어 있는 워크스테이션의 디스크로부터 복사해서 로칼 워크스테이션으로 읽어 옴;
교체될 데이터 블록과 옛 데이터 블록, 그리고 옛 패리티 블록을 XORing을 취함으로 인해 새로운 패리티 블록을 생성;
교체될 데이터 블록과 새로운 패리티 블록 각각을 저장하고 있는 워크스테이션의 2-단계 캐쉬의 MRU 블록으로 위치 (오손 블록 상태로);

2-단계 캐쉬의 오손 블록 교체시 동작은 단순히 디스크에 쓰는 동작만 취하면 된다.

그러면 이러한 환경에서 특정 워크스테이션에 결함이 발생하였을 때 어떻게 동작을 하는지 살펴본다. 우선 결함이 발생한 워크스테이션의 1-단계 캐쉬내의 오손 블록들은 단일 복사 캐싱 기법과 다중 복사 캐싱 기법과 동일하게 복구할 수 없고 단지 옛 블록으로의 복구만 가능하게 된다. 그러나 디스크내의 블록만이 아니라 2-단계 캐쉬내의 오손 블록도 복구가 가능해진다. 2-단계 캐쉬내의 오손 블록은 read-modify-write 동작[7]을 마친뒤의 블록으로 해당 그룹의 패리티 블록이 새롭게 계산된 상태이므로 RAID 시스템에서의 복구 과정을 통해 블록을 복구할 수 있게된다. 과정은 다음과 같다. 복구하고자 하는 블록과 같은 패리티 그룹내에 있는 데이터 블록들과 패리티 블록을 가져와서 XORing 시킴으로 블록을 복구할 수 있게 되는데 블록들을 가지고올 경우 다른 기법에서는 처음부터 디스크에서 가져오는데 본 기법에서는 우선 2-단계 캐쉬를 찾는다. 만약에 2-단계 캐쉬에 없을 경우에 디스크로 부터 해당 블록을 가져오게 되는 것이다. 즉 2-단계 캐쉬부터 검색하고 없을 경우에 디스크로 부터 가져오는 것이다. 그러므로 본 기법에서의 데이터 복구 시간은 다른 기법들에 비해 월등히 감소하게 된다.

Ⅲ. 성능 분석

3.1 성능 분석 방법

성능 분석은 트레이스 기반 시뮬레이션을 통하여 캐싱 기법들을 평가하였다. 성능 분석에 사용된 트레

이스는 스프라이트 트레이스[12]이다. 스프라이트 트레이스는 버클리 대학에서 4대의 파일 서버와 40여대의 클라이언트에서 매일 사용하는 30여명의 사용자와 간헐적으로 사용하는 40여명의 사용자들에게서 얻어진 분산 파일 시스템의 트레이스이다.

본 성능 분석에서는 동일한 파일 서버로부터 여덟 개의 클라이언트 트레이스를 임의로 선택하여 사용하였다. 여덟 개 트레이스들의 전체 요구는 304,181개이다. 트레이스의 접근 형태를 살펴보면 227,767개는 읽기에 대한 요구이고, 76,414개는 쓰기 요구이다. 이러한 트레이스들간에는 동일한 블록에 대한 요구가 포함되어 있어 파일을 공유하는 경우가 존재하며 이들 트레이스들은 시뮬레이터에서 여덟 대의 워크스테이션으로 구성된 분산 환경에서 각 워크스테이션들의 작업부하가 되어 시뮬레이션을 수행하게 된다. 성능 분석에서 각 워크스테이션 캐쉬를 초기화시키는데 소요된 요구를 제외한 나머지 요구를 가지고 평가하였다.

본 성능 분석에서 캐쉬 블록의 크기는 8KB이며, 8KB 데이터에 대한 메모리 접근 시간은 로칼 메모리는 250μs[13], 네트워크 오버헤드는 200μs로 하고 데이터가 네트워크를 통해 전송되는데 걸리는 시간은 400μs[14]로 고정시키고 실험하였으며 특히 디스크 부분은 HP 97560 하드 디스크를 모델링한 시뮬레이터[15]를 사용하였다. 각 기법에 대한 성능 비교 척도는 캐쉬 요구당 평균 응답시간(average response time)과 요구당 시스템 동작시간(average system busy time)이다. 우선 캐싱 기법들에 대해 캐쉬의 크기를 달리하면서 요구당 평균 응답시간을 평가한다. 그러나 소프트웨어 RAID 파일 시스템에서 요구당 평균 응답 시간만을 평가하는 것은 바람직한 측정이 아니다. 왜냐하면 하드웨어 RAID 시스템 뿐만이 아니라 소프트웨어 RAID 파일 시스템에서는 작은 쓰기 동작이 성능에 큰 병목으로 작용하는데 평균 응답 시간에서는 이러한 작은 쓰기 동작으로 인해 발생하는 네 번의 입출력 동작에 대한 측정을 명확하게 할 수 없기 때문에 요구당 시스템 동작 시간을 분석한다. 이는 오손 블록의 교체 발생시 옛 데이터 읽기와 옛 패리티 읽기 동작이 서로 다른 워크스테이션에서 동시에 발생하고 새로운 데이터 쓰기와 새로운 패리티 쓰기 동작 역시 동시에 발생하기 때문에 히트율과 평균 응답 시간에서는 측정할 수 없는 시스템에서의 실질적인 오버헤드를 측정하기 위해서이다. 성능 분석은 여덟 대의 워크스테이션으로 구성된 분산 환경에서 각 워크스테이션 캐쉬의 크기를 변화시키면서 각 정책들의 성능을 분석하였다.

3.2 성능 분석 결과

그림 1과 그림 2는 세 가지의 캐싱 기법들에 대하여 캐쉬의 크기를 달리하면서 블록에 대한 입출력 요구당 평균 응답시간과 요구당 평균 시스템 동작시간을 나타낸 것이다. 그림에서 MCC란 다중 복사 캐싱 기법을 나타내는 것이고 OCC란 단일 복사 캐싱 기법을 나타낸 것이다. 그리고 2LC(70)은 제안한 캐싱 기법에서 전체 캐쉬에서 2-단계 캐쉬가 차지하는 비율이 70%인 경우를 의미하는데 캐쉬 비율에 따른 실험은 그림 3과 그림 4에 나타나있다.

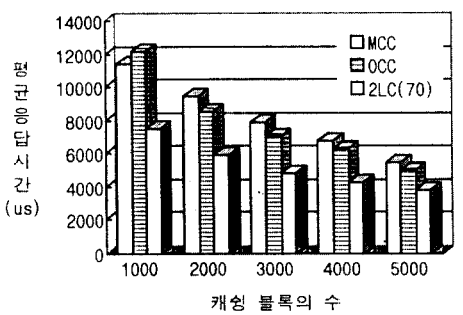

그림 1. 캐싱 기법들에 대한 요구당 평균 응답시간

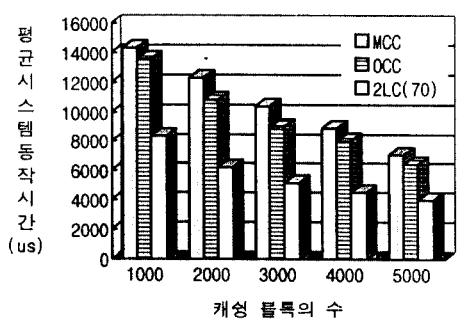

그림 2. 캐싱 기법들에 대한 평균 시스템 동작시간

두 그림 모두 X축은 각 워크스테이션의 캐쉬 블록의 수를 나타내는 것으로 8KB 블록의 개수를 의미한다. 그림 1에서 Y축은 요구당 평균 응답시간을 나타내고 그림 2에서 Y축은 요구당 평균 시스템 동작시간을 의미하는 것으로 단위는 microseconds이다.

그림의 결과에서 몇가지 사항을 관찰할 수 있다. 다중 복사 캐쉬 기법과 단일 복사 캐쉬 기법간에는 성능에 있어서 큰 차이를 보이지 않고 있으나 제안한 2단계 캐쉬 기법은 다른 기법들에 비해 모든 경우에 있어서 평균 응답시간과 시스템 동작시간 모두 좋은 성능을 나타내는 것을 볼 수 있는데 다음과 같은 이유 때문이다. 2단계 캐쉬 기법은 작은 쓰기 동작시 발생하는 네 번의 디스크로의 입출력 요구를 최소화시켜주는 기법으로 작은 쓰기 동작이 발생했을 때 디스크로의 요구가 한 번도 일어나지 않게 해주기도 하기 때문에 다른 기법들에 비해 효율적인 성능을 나타낸다. 이러한 성능상의 차이는 그림에서 볼 수 있듯이 너무도 명확하다. 이러한 결과가 의미하는 바는 다음과 같다. 우선 소프트웨어 RAID 파일 시스템에서는 작은 쓰기 동작이 성능에 매우 큰 병목을 차지한다는 점이고, 본 논문에서 제안한 2단계 캐쉬 기법은 이러한 문제점을 해결하기 위해 제안된 기법으로 시스템 입장에서는 물론이고 사용자 입장에서도 매우 높은 성능 향상을 가져오는 효율적인 기법임을 확인할 수 있다.

그림 3과 그림 4는 2단계 캐쉬 기법에서 2-단계 캐쉬가 차지하는 비율을 달리할 경우의 성능을 나타낸 것이다. 그림에서 70% 가량을 2-단계 캐쉬 영역으로 하는 것이 전반적으로 좋은 성능을 나타내는 것을 볼 수 있는데 다음과 같은 이유 때문이다.

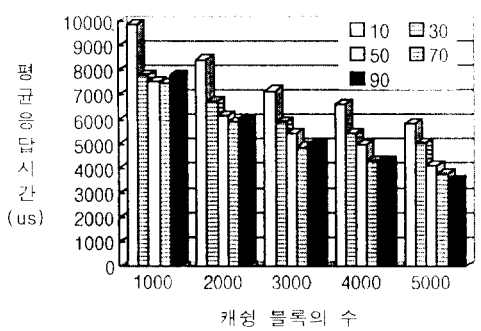

그림 3. 2단계 캐쉬 기법에서 2-단계 캐쉬의 비율에 따른 요구당 평균 응답시간

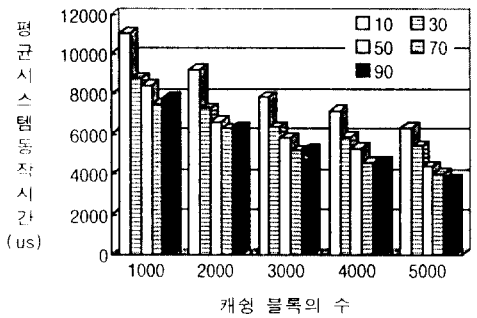

그림 4. 2단계 캐쉬 기법에서 2-단계 캐쉬의 비율에 따른 평균 시스템 동작시간

2-단계 캐쉬 영역은 작은 쓰기 동작시 발생하는 네 번의 디스크로의 요구를 최소화시켜 주는 영역으로 클수록 옛 데이터와 패리티 블록을 캐쉬에 오랫동안 저장하고 있음으로 오손 블록 교체시에 디스크로의 요구를 한 번도 발생하지 않도록 해주기도 하며 또한 2-단계 캐쉬 영역에 저장되어 있는 데이터 블록과 특히 패리티 블록은 빈번히 참조될 가능성이 크기 때문에 2-단계 캐쉬 영역이 차지하는 비율이 다소 클수록 성능이 향상되는 것이다. 그러나 캐쉬에서 2-단계 캐쉬가 차지하는 비율이 80% 이상이 되면 성능이 감소하게 된다. 이는 2-단계 캐쉬 영역은 읽기 요구만 발생한 블록은 저장을 하지 않으므로 활용면에서 어느 정도의 제약을 지니기 때문이다.

Ⅳ. 결 론

소프트웨어 RAID 파일 시스템이란 네트워크로 연결된 다수의 시스템들이 보유한 각 디스크를 통해 하드웨어 RAID의 기능을 소프트웨어적으로 제공하는 시스템을 의미한다. 이러한 시스템은 기존의 분산 파일 시스템에 비하여 높은 성능과 신뢰성을 제공해준다. 본 논문에서는 소프트웨어 RAID 파일 시스템에서 효율적인 캐쉬 기법을 제안하였다. 그리고 이와 기존 분산 파일 시스템의 캐쉬 기법을 소프트웨어 RAID 파일 시스템에 적용한 두가지 기법들과의 성능을 비교하였다.

하드웨어 RAID 시스템에서와 마찬가지로 소프트웨어 RAID 파일 시스템에서도 작은 쓰기 동작은 성능을 크게 저하시키는 요소임을 확인할 수 있었다.

이러한 문제를 해결하기 위해 캐쉬의 영역을 논리적인 두 단계로 구성하였다. 2-단계 캐쉬 영역에 옛 데이터와 패리티 정보를 유지시킴으로 오손 블록 교체시에 발생하는 네 번의 디스크에 대한 요구를 최소화시켜 준다. 이러한 캐싱 기법들에 대한 성능 비교는 트레이스 기반 시뮬레이션에 의해 수행되었다. 실험을 통해 본 논문에서 제안한 2단계 캐싱 기법이 다른 두가지 기법들에 비해 시스템 입장은 물론이고 사용자 입장에서도 매우 효율적인 성능을 나타내는 기법임을 확인할 수 있었다.

참 고 문 헌

1. Russel Sandberg et al, "Design and Implementation of the SUN Network File System," In *Proceedings of the Summer 1985 USENIX Conference*, pages 119-130, Portland, Oregon, 1985.
2. J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch, "The Sprite Network Operating System," *IEEE Computer*, 21:23-36, February 1988.
3. J. H. Hartman and J. K. Ousterhout, "The Zebra striped network file system," *ACM Transactions on Computer System*, 13(3):274-310, August 1995.
4. T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang, "Serverless Network File System," *ACM Transactions on Computer Systems, 14(1):41-79*, February 1996.
5. T. E. Anderson, D. E. Culler, and D. A. Patterson, "A Case for NOW(Networks of Workstations)," *IEEE Micro*, 15(1):54-64, February 1995.
6. 김 종훈, 노 삼혁, 원 유헌, "NOW(Network of Workstations) 환경에서 소프트웨어 RAID 파일 시스템의 설계 및 성능 평가", 한국통신학회 논문지, 제22권 6호, 1997년 6월.
8. M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson, "Cooperative Caching: Using remote client memory to improve file system performance," In *Proceedings of the First Symposium on Operating System Design and Implementation*, pages 267-280, November 1994.
7. P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys*, 26(2):145-185, June 1994.
8. Daniel Stodolsky, Mark Holland, William V. Courtright III, and Garth A. Gibson, "Parity-logging Disk Arrays," *ACM Transactions on Computer Systems*, 12(3), pp. 206-235, August 1994.
9. Jai Menon, Janes Roche, and Jim Kasson, "Floating Parity and Data Disk Arrays," *Journal of Parallel and Distributed Computing*, pp. 129-139, January 1993.
10. Jai Menon and Jim Cortney, "The Architecture of a Fault-tolerant Cached RAID Controller," In *Proceedings of the 20th Annual International Symposium on Computer Architecture(ISCA '93)*, pp. 76-86. May 1993.
11. A. Leff, J. L. Wolf, and P. S. Yu, "Efficient LRU-Based Buffering in a LAN Remote Caching Architecture," *IEEE Transactions on Parallel and Distributed Systems*, 7(2):191-206, February 1996.
12. M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout, "Measurements of a Distributed File System," In *Proceedings of the ACM Thirteenth Symposium on Operating Systems Principles*, pages 198-212, October 1991.
13. R. Martin, "HPAM: An Active Message Layer for a Network of HP Workstations," In *Proceedings of the 1994 Hot Interconnects II Conference*, 1994.
14. K. K. Keeton, T. E. Anderson, and D. A. Patterson, "LogP Quantified: The Case for Low-Overhead Local Area Networks," In *Proceedings of the 1995 Hot Interconnects III Conference*, 1995.
15. David Kotz, Song Bac Toh, and Sriram Radhakrisshnan, "A Detailed Simulation Model of the HP 97560 Disk Drive," *Technical Report PCS-TR94-220*, Dartmouth College, Computer Science, Hanover, NH, 1994.

김 종 훈(Jong-Hoon Kim) 정회원
한국통신학회 논문지 제22권 6호 참조

노 삼 혁(Sam H. Noh) 정회원
한국통신학회 논문지 제22권 6호 참조

원 유 헌(Yoo-Hun Won) 정회원
한국통신학회 논문지 제22권 6호 참조