2001년 한국정보처리학회 추계 학술발표논문집 제8권 제2호

병렬 파일 시스템에서 이중 캐쉬 구조

장원영*, 김재열**, 서대화*
*경북대학교 전자공학과
**한국전자통신연구원
e-mail:bisan74@palgong.knu.ac.kr

Dual-Cache Scheme in Parallel File System

Won-Young Jang*, Chei-Yol Kim**, Dae-Wha Seo*
*Department of Electronics, Kyungpook National University
**Linux Research Team, ETRI

요약

프로세스와 디스크 입출력 속도를 비교해보면, 디스크 입출력의 속도가 훨씬 더 느리다. 따라서, 디스크 입출력은 현재의 컴퓨팅 환경에서 병목현상이 되고있다. PFSL(Parallel File System for Linux)은 이런 문제를 해결하기 위한 클러스터링 환경의 병렬 파일 시스템이다. PFSL은 리눅스 머신 상에서 POSIX 스레드 라이브러리를 이용하여 멀티 스레드로 수행된다. 이 논문에서는 PFSL의 성능을 개선하기 위해 클러스터 환경의 작업 부하에 적합하도록 설계한 이중 캐쉬 구조를 소개하고자 한다.

1. 서론

최근에 등장한 응용프로그램들 중에는 단일 프로세서가 제공할 수 있는 계산 능력의 몇 배에 해당하는 능력을 요구하는 것들이 많이 있다. 이 때 프로세서의 한계를 극복하기 위해 효과적인 방법은 여러 개의 프로세서를 함께 연결해서 서로 협력하도록 하는 것이다. 클러스터링은 현재 이런 협력을 수행하기 위한 대표적인 방법이다. 클러스터링을 통해 계산의 규모가 커지고, 취급하는 데이터의 양이 증가함에 따라 큰 데이터를 효율적으로 처리하기 위한 다양한 기법들이 연구되고 있다.

현재 가장 큰 문제점으로 지적되고 있는 것은 대용량의 데이터를 저장하고 읽어 올 때 프로세스 속도에 비해서 디스크의 입출력 속도가 느리기 때문에 병목현상이 발생하게 되는 것이다. 이 입출력 문제를 해결하기 위해서는 입출력 시스템에 병렬성을 확장하는 것이 효과적이며, 병렬 파일 시스템을 통해 입출력 병렬성을 구현할 수 있다.

병렬 파일 시스템에서 대용량 파일의 데이터 블록들은 많은 파일 서비스 노드들에 분산되어 있다. 따라서, 클라이언트가 파일에 접근하려고 할 때, 파일 서비스 노드들은 동시에 접근이 된다. 따라서 입출력 작업부하가 입출력 대역폭을 높이기 위해 여러 대의 서버에 분산되는 것이다.

병렬 파일 시스템 성능을 좌우하는 중요한 요인은 데이터 블록의 재사용 함으로 디스크 입출력 횟수를 줄여주는 캐싱 정책을 꼽을 수 있다. 본 논문에서는 리눅스 클러스터 환경에서 동작하도록 구현된 PFSL (Parallel File System for Linux)[5]에 본 논문이 제안하는 이중 캐쉬 구조(Dual-Cache Scheme)를 적용하여 그 성능을 평가하고 분석하였다.

본 논문은 서론에 이어 2장에서 병렬 파일 시스템에서 이중 캐쉬 구조와 관련이 있는 연구를 살펴보고, 3장에서는 본 논문에서 제안하는 이중 캐쉬 구조가 기반으로 하는 병렬 파일 시스템의 구조에 대해서 알아본다. 4장에서는 본 논문에서 제안하는 이중 캐쉬 구조에 대하여 구체적으로 설명한다. 5장에서는 실험 환경에 대한 내용과 실험을 통해 나온 결과를 고찰하고 마지막으로 6장에서 결론을 제시한다.

2. 배경 연구

병렬 파일 시스템은 파일 입출력 시스템의 대역폭를 확장해 주기 위한 해결책이다. 이러한 병렬 파일 시스템에는 현재 GPFS(Galley parallel file system)[1]와 PVFS (Parallel Virtual File System)[7] 등이 있다.

GPFS는 1996년 Darthmouth 대학에서 개발된 것으로서 NOW(Networks Of Workstations)[2]상에서 동작하도록 디자인 되었으며 현재는 IBM RS/6000과 IBM SP-2 병렬 슈퍼컴퓨터 상에서 운영되고 있다.

PVFS는 PC 클러스터 시스템을 위한 병렬 파일 시스템을 제공하려는 것으로서 PVFS는 글로벌 이름 공간(global name space), 스트라이핑, 다중 사용자 인터페이스 등을 제공한다.[7]

디스크에서 한번 읽어들인 데이터 블록을 재사용함으로써 디스크 입출력 횟수를 줄이는 캐싱에 대한 연구[3][4]도 많이 이루어지고 있으며, LRU(Least Recently Used)와 Segmented LRU(Least Recently Used)[3], LFU (Least Frequently Used)[4] 등이 있다.

LRU(Least Recently Used) 교체정책은 캐쉬 내에 최근에 사용되지 않았던 데이터 블록을 교체시키는 것으로 다양한 종류의 작업부하 상에서 뛰어난 성능을 발휘한다. 그러나 오디오, 비디오 파일처럼 대체로 대용량이면서 순차적으로 접근되어 파일의 블록들이 처음 참조된 후에는 결

코 참조되지 않는 경우에는 캐시의 적중률이 낮아져 캐싱의 효과가 없으며, 재사용되지 않는 데이터의 캐싱로 인해 시스템 전체의 성능을 저하할 수도 있다.

Segmented LRU(Least Recently Used) 교체정책은 LRU의 확장 정책으로써 한 번 이상 사용된 데이터 블록이 다시 사용될 가능성이 높다는 것에 기반한 정책이다.[3] 즉, 한 번 이상 요청된 파일 블록들은 가능하면 교체가 되지 않도록 한다.

LFS(Least-Frequently Used) 교체정책은 버퍼 캐쉬 블록의 참조 회수를 기준으로 교체할 대상을 결정한다. 즉 각각의 블록들은 참조회수를 기록하는 카운터를 가진다. 교체할 대상이 되는 블록은 가장 작은 참조 회수를 가지는 블록이 선택된다. LFU 교체정책은 실제로 구현하기에는 적당하지 않으므로 기본 알고리즘을 변형시킨 알고리즘이 제안되었다.[4]

이 논문에서 제안된 방법은 순차적인 파일 접근 패턴에 대하여 작업부하를 고려한 이중 캐쉬 구조를 이용함으로 입출력 성능을 향상시킬 수 있다.

3. 클러스터 시스템에서 병렬 파일 시스템

PFSL 은 리눅스 클러스터 시스템을 위한 병렬 파일 시스템이다. 이것은 File Servers(FSs), block servers(BSs), 그리고 metadata server(MS)로 구성된다(그림 1). FS는 프로세스 노드에서 파일 수준의 파일 handling을 위한 인터페이스를 제공하고 블록 서버는 입출력 노드에서 분산 저장된 파일에 대한 데이터 블록의 입출력 서비스를 담당하며 메타 데이터 서버는 분산 저장된 전체 파일에 대한 메타 데이터를 관리하고 작업이 옳게 되고 있는지 확인하기 위해 정기적으로 블록 서버를 체크한다.

응용프로그램이 PFSL 라이브러리를 통해 FS에게 파일에 대한 서비스를 요청하면, FS는 응용프로그램의 요구를 수행하기 위해 MS에 있는 메타 데이터를 이용하여 필요한 데이터 블록의 논리적 주소를 연산한 후, 입출력 노드의 BS에게 데이터 블록에 대한 서비스를 요구한다. 메타 데이터는 파일 사이즈, 파일 허가권, 소유권 등과 같은 평범한 파일 시스템 정보와 함께 블록 크기, BS 리스트 등과 같은 추가적인 정보를 포함하고있다.

큰 사이즈의 파일은 BS에 의해 저장된 블록으로 나누어진다. BS는 역시 데이터 블록을 위한 버퍼 캐쉬를 가진다. 즉, PFSL은 두 가지 레벨(FS side and BS side)의 캐쉬를 가진다.

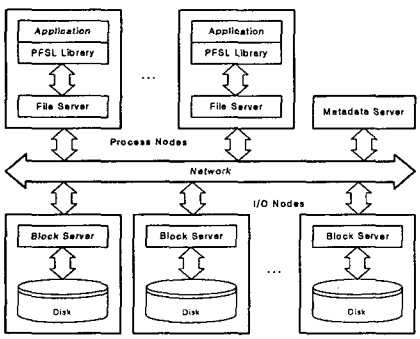

Figure 1 : PFSL의 구성

BS는 FS로부터 데이터 블록에 대한 서비스 요청이 있을 경우, 디스크의 입출력을 통해 데이터 블록의 읽기/쓰기 요청을 서비스한다.

FS에서 멀티쓰레드를 이용해서 동시에 여러 BS로 데이터 블록을 요구하여, BS에서의 데이터 블록 서비스가 병렬로 진행될 수 있다. 이러한 병렬 처리의 장점은 입출력 노드가 많아질수록 데이터의 입출력에 대한 성능 향상을 가져온다.

4. 이중 캐쉬 구조

4.1 이중 캐쉬 구조의 기본 개념

파일 접근 패턴을 보면, 파일의 시작부분을 소비하지 않고 파일이 사용되는 경우는 거의 없으며, 이전의 연구들은 대부분의 파일 액세스가 파일의 시작부터 끝 부분까지 순차적으로 이루어짐을 보여주고 있다[6][9]. 이 결과는 파일의 시작 부분과 나머지를 동등하게 보는 것이 적절하지 않음을 암시하는 것이다.

따라서, 전체 파일 중 파일의 시작 부분만을 따로 캐쉬에서 관리함으로써 캐싱의 성능을 개선할 수 있다. 이런 생각에 기인해서, 본 논문에서는 캐쉬를 두 부분 즉, FPC(Front -Part Cache)와 RPC(Residual-Part Cache)로 나누었다. FPC는 각 파일의 최초의 몇 개의 블록들만을 캐싱하는 반면, RPC는 선반입될 블록을 포함한 나머지 블록을 캐싱한다.

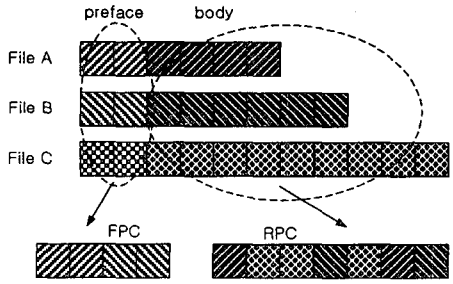

그림 2 : DCS 구조

FPC는 RPC에 비해 크기를 작게 잡았다. 하지만 파일의 크기가 동일하지 않더라도 파일 preface의 크기가 파일 body에 비해서 작은 부분을 차지하게된다. 따라서, 파일의 preface가 FPC에 있을 확률이 파일의 body가 RPC에 있을 확률보다 훨씬 높게 된다.

FPC내에 캐싱된 블록의 숫자는 총 캐쉬 사이즈, 각 파트의 비율, 파일의 수, 응용프로그램의 성격에 따라서 조정될 수 있으며, 이것이 작업부하를 고려한 이중 캐쉬 구조이며 파일 시스템 성능을 보다 향상시킬 수 있다.

4.2 작업부하를 고려한 이중 캐쉬 구조

파일의 preface부분의 크기를 현재 작업 부하를 반영하여서 동적으로 결정하여, FPC에 캐싱하도록 하였다. 예를 들어, 파일 시스템 내에 100MB의 파일이 10개 있을 때와 10MB 파일이 100개 있을 때, 같은 크기의 파일 preface가 양쪽의 경우에 적용된다면 동일한 성능을 내지는 못할 것이다. 따라서, 작업부하를 고려한 이중 캐쉬 구조는 이런 문제를 해결하기 위해 제안되었다. 이 구조는 FPC내에서 고정된 파일 숫자를 동적으로 조정하고, 또한 현재 파일

요청 상황에 따라 파일 preface 크기를 조정하는 것이다.

아래는 작업부하를 고려한 이중 캐쉬 구조의 알고리즘을 설명하기 위한 용어들이다.

S_{tc} : 총 캐쉬 크기

S_{fp} : FPC의 크기

S_{pof} : 파일 preface의 크기

N_f : 활성 파일의 수

S_{tf} : 활성 파일들의 총 크기

이때, 단일 캐쉬 구조에서 활성 파일이 캐쉬에 있을 확률 $P_s = S_{tc}/S_t$ 이며, 활성 파일의 preface가 FPC에 있을 확률 $P_{pof} = S_{fp} / (N_f \cdot S_{pof})$이 된다.

병렬 파일 시스템에서 사용되는 파일들은 일반적으로 단일 프로세서 환경에서 사용되는 파일들에 비해 상대적으로 크기가 크며 파일이 열렸다 닫힐 때까지의 시간, 즉 파일이 사용되는 시간이 길어 현재 열려진 파일의 개수를 세는 작업은 의미가 있다.

작업부하를 고려한 이중 캐쉬 구조는 파일의 preface 크기를 현재의 파일 상황에 따라 동적으로 변화시킨다. 즉 요청된 파일의 개수가 많을수록 파일 preface의 크기를 줄이고 요청된 파일의 개수가 적을 경우에는 파일 preface의 크기를 늘인다.

그러나 이러한 경우에 파일 preface의 크기가 효과가 없을 정도로 줄어들 수 있으며, 어떤 경우에는 파일 preface의 크기가 과도하게 커져서 파일 시스템 성능에 도움이 되지 않을 수도 있다. 본 시스템에서는 이러한 경우를 방지하게 위하여 파일 preface의 크기에 상한선과 하한선을 설정하였다. 이렇게 함으로써 파일 요청이 동시에 일어날 때 파일 preface의 서두가 극도로 줄어드는 것을 막을 수 있고 파일 요청이 한가한 경우 파일 preface의 크기가 극도로 커지는 경우를 막을 수 있다.

병렬 파일시스템의 특징을 활용하기 위하여 선반입하는 data의 양은 BS의 개수와 블록 크기의 곱의 배수가 되도록 하여, 모든 BS에서 동시에 한 블록이상 하도록 하였다. 또한, FPC 내에서 buffer가 모자라서 현재의 블록들을 교체하여야 하는 경우에는 일반적인 캐쉬에서 블록 단위로 교체하는 것이 아니라 파일 단위로 파일 preface를 교체한다. 이렇게 함으로써 블록 단위로 교체할 때의 과부하를 줄여 좀 더 빨리 FPC의 적응성을 높일 수 있다.

$P_{pof} = \alpha P_s$ (α: fixed value, $\alpha > 1$)

$S_{fp} = P_{pof} \cdot N_f \cdot S_{pof} = \alpha P_s \cdot N_f \cdot S_{pof}$

$S_{pof}(min) \leq S_{pof} \leq S_{pof}(max)$

α는 파일 preface의 크기를 조절하기 위한 값이다. 위 수식에서의 N_f 와 P_s , P_{pof} , S_{fp} 의 값들은 시스템 상황에 따라 정해지며, 본 논문의 실험 결과에서 α의 변화에 따른 평균 응답시간에 의해 α는 가장 이상적인 값으로 정해진다. 이런 값들에 의해 파일 preface의 크기(S_{pof})는 동적으로 조정되게 된다.

5. 실험

5.1 실험 환경

실험 환경은 4대의 컴퓨터로 구성되었다. 각 컴퓨터는 600MHz의 CPU와 128Mb의 메인 메모리, 20Gb의 하드디스크를 가지고 있으며, 100Mbps의 인텔 NIC를 장착하고 있다. 이 4대의 컴퓨터는 100Mbps 속도의 인텔사 허브에 연결되어 있으며 다른 네트웍 트래픽의 영향을 제거하기 위해 테스트 베드만을 연결하였다. 각 노드는 리눅스를 운영체제로 하고 있으며 커널의 버전은 2.2.16이다.

5.2 작업부하 테스트

실험에 사용된 작업부하는 기존 논문의 작업 부하 연구[2][4][8]를 기반으로 하였으며, 응용프로그램에서 작업 부하를 발생시키도록 하였다. 실험에서 사용된 작업부하는 아래의 표와 같다.

Table I. 작업부하 특성

작업부하	파일의 수	평균파일 크기(MB)	순차적인 요청율(%)	전체 파일 요청율(%)	평균요청 크기(kbps)
A	20	10	70	60	500
B	40	5	70	60	200
C	20	10	50	40	200
D	40	5	50	40	50

$$\text{평균 파일크기} = \frac{\sum \text{파일크기}}{\text{파일들의 수}}$$

$$\text{평균 요청크기} = \frac{\sum \text{요청크기} / \text{각 요청간 시간 간격}}{\text{파일들의 수}}$$

$$\text{순차적인 요청} = \frac{\text{순차적인 접근 파일들}}{\text{파일들의 수}}$$

Table I에서의 파일들은 입출력이 요청된 파일들을 말하며 활성 파일이라고 한다. 평균 파일 크기는 파일 크기들의 합에 파일들의 수를 나눈 것이다. 평균 요청크기는 각 요청간 시간 간격과 요청크기를 파일의 수로 평균을 낸 것이다. 순차적인 요청은 총 파일에서 순차적으로 접근되는 파일들의 비율이며 전체파일 요청율은 파일을 처음부터 읽어서 끝까지 사용되는 비율이다.

각 작업 부하는 서로 다른 특성들을 가지고 있다. 작업부하 A는 작업 부하 B, D보다 파일의 수는 적지만 더 큰 파일들을 가지고 있으며 순차적인 요청율이 더 높고 파일을 처음부터 읽어서 끝까지 사용되는 것이 대부분인 작업부하이다.

작업부하 A의 경우에 이중 캐쉬 구조를 적용할 때 더욱 효과적이며 다른 상태에 있는 작업 부하와 비교하여 실험하였다. 그 실험 결과에 의해 파일 preface들의 크기를 조정하는 α의 값을 결정한다.

5.3 실험 결과

실험에 사용된 작업 부하의 전체 파일의 합은 200MB 정도가 된다. 이를 감안해 캐쉬 전체의 크기를 10MB로 고정 시켰다. 작은 파일의 대한 캐쉬 크기와 성능에 대한 연구는 많았으므로[2][8], 본 실험에서는 전체 파일 크기에 비해서 적당한 캐쉬 크기를 잡고 이에 대해서 실험을 하였다. 실험에 사용된 전체 캐쉬 크기는 10MB로 이중 FPC 크기를 1MB로 고정시켰다. 일반적으로 하나의 프로세스에서 한번에 요청하는 데이터 양은 물리적 메모리 용량의 한계 때문에 그다지 크지 않다. 이에 따라 한 파일당 FPC에서 할당받을 수 있는 최대 크기를 10 블록으로 제한시켰다. 이렇게 파일 preface의 크기를 제한하는 이유는 앞에서 설명한 바와 같다.

그림 4는 단일 캐쉬 구조와 이중 캐쉬 구조에서의 캐쉬의 적중률을 비교한 결과이다. 아래의 결과에서 단일 캐쉬 구조보다 이중 캐쉬 구조에서 적중률이 조금 높아짐을 알 수 있다. 이것은 이중 캐쉬 구조를 적용하여 파일 preface에 대한 적중률이 증가하였기 때문으로 전체 캐쉬 적중률이 FPC에 대한 적중률만큼 증가하게 된 것이다.

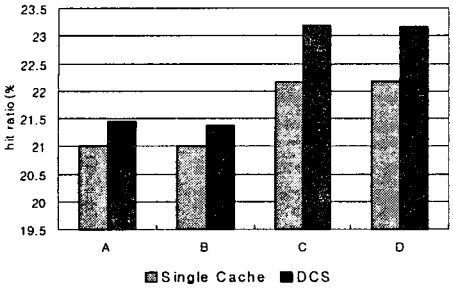

그림 4 : 단일 캐쉬와 이중 캐쉬 구조의 적중률

그림 5는 α값의 변화에 따른 평균 응답 시간을 나타낸 것이며, 그래프에서 성능의 차이가 확연히 드러나게 된다. 이것은 파일 요청에 대한 응답 시간을 향상시키기 위해서는 많은 파일들의 preface가 캐시에 존재하는 것이 유리하다는 것을 의미한다. 본 실험에 사용된 시스템에서는 α의 값을 20으로 고정시켜 사용하는 것이 효율적이다.

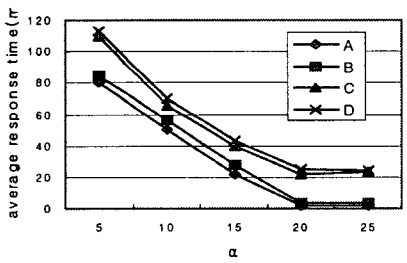

그림 5 : α의 평균 응답시간

위의 결과를 분석해 보면 사실 α의 값이 20 근처에서 최상의 성능을 나타낸다. 즉, 파일 preface가 캐시에 있을 확률을 파일 body가 RPC에 있을 확률보다 20배 정도 크게 했을 때가 적당하다는 결과이다.

파일 preface가 꼭 20이 될 필요는 없지만 현재의 캐쉬 크기와 파일의 전체 크기에 비추어 가장 적당한 것이라는 것을 알 수 있다. 즉, 지금의 α는 접근하는 파일의 전체 크기가 정해져 있는 특수한 경우의 최선의 값이므로 이는 파일 시스템 내에 저장된 전체 파일에 대한 상황 또한 감안해야 한다는 것을 알 수 있지만 이번 실험에서는 이점은 배제되었다.

6. 결론

캐싱 정책은 워크스테이션과 PC상에서 일반적인 입출력 부하를 줄이는데 아주 효과적이다. 하지만, 그것을 큰 파일을 다루는 병렬 파일 시스템에 그대로 적용하기에는 부적합하다. 병렬 파일 시스템에서는 파일이 캐쉬 내에 있는 시간과 캐쉬를 액세스하는 시간이 매우 길기 때문에 참조 특성들이 일반 워크스테이션이나 PC와는 아주 다르기 때문이다[6].

이 논문에 제시된 이중 캐쉬 구조는 병렬 파일 시스템이나 클러스터에 사용되는 큰 파일에 적용할 때 아주 효과적이며, 캐쉬 적중률이 많이 증가되지 않지만 파일요청에 대한 응답시간을 줄일 수 있다.

참고문헌

[1] N. Nieuwejaar and D. Kotz, "The Galley Parallel File System," Proc. of the 1996 international conference on Supercomputing, PP. 374 - 381 May 1996

[2] T. Anderson, D. Culler, and D. Patterson, "A Case for Networks of Workstations," IEEE Micro, Feb. 1995, http://now.cs.berkely.edu/

[3] R. Karedla, J. S. Love and B. G. Wherry, "Caching strategies to improve disk system performance," IEEE COMPUTER, pp. 38-46, March 1994.

[4] J. T. Robinson and M. V. Devarakonda, "Data cache management using frequency-based replacement," In proc. of the conference on Measurements and Modeling of Computer Systems, ACM press, May 1990, pp. 134-142.

[5] A. Purakayastha, "Chracterizing and Optimizing and Parallel File Systems," Dissertation of Duke University, Durham, N.H. pp. 1-10, June 1996.

[6] A. Purakayastha, C. S. Ellis, D. Kotz, N. Nieuwejaar and M. Best, "Characterizing Parallel File-access Patterns on a Large-scale Multiprocessor," In Proc. of the Ninth International Parallel Processing Symposium, pp. 165-172, April 1995.

[7] W. B. Ligon III, and R. B. Ross, "An Overview of the Parallel Virtual File System," Proc. of the 1999 Extreme Linux Workshop, June 1999.

[8] Drew Roselli, Jacob R. Lorch, and Thomas E. Anderson, "A Comparison of File System Workloads," Proc. of the USENIX Annual Technical Conference, San Diego, CA, June 2000.

[9] Evgenia Smirni and Daniel A. Reed, "Workload Characterization of Input/Output Intensive Parallel Applications, " Proc. of the Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Springer-Verlag Lecture Notes in Computer Science, June 1997, V

[10] J. Ousterhout, H. D. Costa, and D. Harrison, J. Kunze, M. Kupfer, and J. Thompson, "A Trace driven analysis of the UNIX 4.2 BSD file system," Proc. of the Tenth ACM Symposium on Operating Systems Principles, pp. 15-24, Dec. 1985.