2000년 한국정보처리학회 추계 학술발표논문집 제 7 권 제 2 호

병렬파일 시스템에서 I/O 대역폭 개선을 위한 이단 선반입 기법

황보준형 , 조종현 , 이윤영 , 서대화
경북대학교 전자공학과
e-mail : bluesky@palgong.knu.ac.kr , alfcom@palgong.knu.ac.kr
yylee@palgong.knu.ac.kr , dwseo@ee.knu.ac.kr

Two-level prefetching method for I/O bandwidth enhancement in Parallel File System

Jun-Hyung Hwang-Bo , Jong-Hyun Cho , Yoon-Young Lee , Dae-Wha Seo
Dept. of Electronic Engineering , Kyungpook National University

요 약

병렬 파일 시스템은 늦은 디스크 I/O 로 인한 성능 저하를 개선하기 위해 병렬 I/O 를 제공한다. 이 때 계산과 디스크 I/O 를 중첩 시키는 선반입 기법으로 디스크 I/O 로 인한 성능 저하를 더욱 개선할 수 있다. 하지만 I/O 위주의 프로그램에서는 선반입으로 인하여 시스템에서 제공하는 I/O 대역폭을 넘어 최악의 경우 기존의 선반입 기법은 성능개선을 위한 최선이 될 수 없을 뿐 아니라 선반입 기법 자체가 과부하가 될 수 있다. 본 논문에서는 이런 상황을 고려하여 I/O 대역폭 개선을 위한 이단 선반입 기법을 제시하여 성능개선을 제공한다.

1. 서론

지난 몇 년간 컴퓨터기술의 발전은 프로세서 중심으로 비약적인 발전을 거듭하고 있지만 디스크 저장장치는 용량의 발전에 비해 입출력 속도의 발전이 프로세서 발전 속도를 따라가지 못하고 있다. 점차 대용량의 데이터 액세스를 원하는 시점에서 디스크 입출력 대역폭의 문제는 아주 심각하게 대두되고 있다. 이러한 이유로 디스크 입출력 속도의 문제를 해결하고자 하는 연구가 많이 이루어 지고 있다.

병렬 파일 시스템은 입출력 작업 부하를 다중 서버로 분산 시켜 입출력 대역폭을 확장하며, 실행시간을 단축 시켜 성능을 향상 시키는 소프트웨어적인 입출력 문제 해결 방법이다[3]. 여기에 선반입 기법을 추가하여 필요한 데이터를 미리 읽어옴으로써 디스크 입출력으로 인한 시간 지연을 줄일 수도 있다.

하지만 기존의 병렬 파일 시스템에서는 프로세서가 연속된 데이터를 요구하는 속도가 빠른 경우 단순히 캐쉬미스를 줄이고자 하는 선반입 기법이 응용프로그램의 Computation 시간동안 완료되지 못하고 다음 읽기 요청과 중첩된다면 시스템에서 제공하는 입출력대역폭을 넘어서서 오히려 시스템의 성능저하를 가져올 수 있다.

본 논문에서는 이런 성능저하를 개선하는 방법으로 파일 서버와 블록 서버에서 이루어 지는 이단 선반입 기법을 제안한다.

이단 선반입 기법은 블록서버 쪽에서 추가적으로 이루어지는 선반입 기법으로 블록서버 쪽의 디스크로부터 블록서버 로컬캐시로 미리 데이터를 가져온다. 이는 파일서버 선반입과는 비동기적으로 이루어 진다. 이 블록서버의 선반입 기법으로 파일서버 선반입 시간을 줄일 수 있다. 이는 파일서버 선반입과 읽기 요청의 중첩으로 인한 시스템 성능 저하를 개선할 수 있다.

I/O 위주의 프로그램에서 최소한의 선반입 시간이라도 시스템에서 제공해주는 대역폭을 넘어 시스템에 과부하를 발생시킨다면 파일서버의 선반입을 중지 하는 방법을 이용하여 성능 향상을 이끌어 낼 수도 있다.

본 논문은 2 절에서 병렬 파일 시스템과 데이터 선반입 기법들에 대해 살펴보고, 3 절에서는 제안하는 선반입 기법을 서술하며, 4 절에서는 제안한 선반입 기법

에 대한 실험과 결과 분석, 마지막으로 5 절에서는 결론을 맺는 것으로 구성된다.

2. 관련연구

2.1 선반입 알고리즘 기법

데이터 선반입 기법은 앞으로 사용될 데이터를 프로세서가 요구하기 전에 미리 읽어오는 방법이다. 이 선반입 기법은 캐쉬의 미스를 줄여 주므로 성능 향상에 중요한 역할을 한다.

이런 이유로 여러 가지 선반입 기법이 연구되었다. 그 중 파일의 액세스 패턴이 순차적이라는 가정을 바탕으로 하는 OBA(One-Block Ahead) 방식과 이를 개선한 aggressive 방식이 있다. 이들은 현재 읽혀진 위치를 기준으로 다음 데이터 블록을 가져오는 방식이다. 이들은 구조가 단순하다는 장점이 있다.

또한 데이터 압축을 기반으로 한 응용 프로그램이 정규적인 형태로 파일을 액세스 한다는 사실을 바탕으로 한 ISG(Interval-and-Size Graph)방식의 기법이 있다. 이는 일단 파일이 액세스 되면 그 형태가 액세스되는 블록들 간의 간격 및 크기 등의 정보를 기반으로 다음에 사용될 블록들을 예측하는 기법이다[4].

하지만 이 모든 선반입 기법은 캐쉬의 미스를 줄이고자 하는데 그 목적을 두고 있다. 이 기법들은 사용자측에서 과도한 I/O 를 요구한다면 선반입 액세스 시간 때문에 현재의 읽기 요청이 지연되어 성능향상은 기대할 수 없을 뿐 아니라 최악의 경우 엄청난 과부하가 될 수도 있다.

이를 해결하고자 선반입의 양을 조절하는 순차적 선반입 기법들이 Tcheun[1]에 의해 제안되었다. 하지만 이 기법도 테이블 등 선반입 조절을 위한 부가적인 구현이 필요하여 시스템이 복잡해지는 단점이 있다.

2.2 기존 선반입 알고리즘 구조

기존 병렬 파일 시스템에서의 선반입 구조는 응용프로그램이 계산을 하는 동안 읽기 요청이 없을 시에 선반입 쓰레드를 생성한다. 이 선반입 쓰레드는 디스크와의 통신을 통해 다음에 읽혀질 블록을 미리 로컬 캐시에 가져오게 된다. 그럼 다음 읽기 요청 때 메인 쓰레드는 먼저 로컬 캐쉬를 탐색하게 된다. 이때 필요한 블록이 이미 파일서버 로컬캐쉬에 있기 때문에 디스크 액세스 시간을 없앨 수 있다. 결과적으로 읽기 액세스 시간을 줄일 수 있게 된다.

하지만 충분한 선반입 시간이 보장되지 않아 선반입 도중에 read 요청이 응용프로그램에서 들어온다면 시스템에서 제공되는 입출력 대역폭을 넘어서는 I/O 로 인해 이상적인 성능향상을 기대할 수 없게 된다.

3. 이단(Two-level) 선반입 기법

PFSL(Parallel File System for Linux)[2]은 응용프로그램에서 요청을 받아 파일 시스템 서비스를 해주는 파일서버와 파일서버의 요청을 받아 디스크에 데이터를 쓰거나 읽어 들이는 블록서버로 구성된다.

파일서버는 응용프로그램에서의 읽기/쓰기 요청을 담당할 쓰레드를 블록 서버의 개수 만큼 생성한다. 이렇게 생성된 쓰레드는 블록서버와 통신을 하면서 응용프로그램의 요청을 처리하게 된다.

블록서버는 파일서버에서 온 명령을 처리할 쓰레드를 생성한다. 이렇게 생성된 쓰레드는 디스크와 액세스하면서 파일서버의 요청을 처리하게 된다.

이단 선반입 기법은 기존 병렬 파일 시스템의 선반입으로 소요되는 액세스 시간을 줄여서 입출력 대역폭을 개선해준다. 파일 서버의 선반입과 함께 블록 서버에서도 POSIX 쓰레드 라이브러리에서 제공하는 쓰레드를 이용하여 이단 선반입을 한다.

블록서버에서는 파일서버에서 요구한 블록과 더불어 추가로 쓰레드를 이용하여 요청된 블록의 다음 블록을 블록서버의 디스크에서 읽어온다. 파일 서버에서 요청된 블록들 외에 몇 개의 블록을 더 읽어 블록 서버쪽 로컬 캐시로 갔다 놓아도 블록 서버쪽에서는 특별한 부하가 되지않는다. 이런 이유로 파일 서버 선반입 측면에서 보면 미리 가져올 데이터가 디스크가 아니라 블록 서버쪽 로컬 캐시에 있어 디스크 액세스 시간을 없앨 수 있고, 전체적으로 선반입 액세스 시간을 줄일 수 있게 된다. 그 결과 파일서버에서는 선반입과 읽기 요청을 최대한 중첩 시키지 않을 수 있다.

3.1 파일서버 선반입 구조

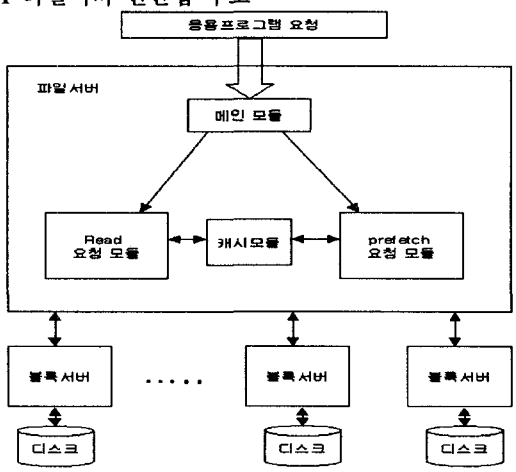

그림 1 파일서버 선반입 구조

파일서버는 응용프로그램에서 오는 요청을 처리하는 파일서버 메인 모듈과 읽기 요청을 담당하는 read 요청 모듈, 선반입을 위한 prefetch 요청 모듈로 구성된다<그림 1>.

파일서버의 메인 모듈은 응용 프로그램에서 오는 요청을 담당할 모듈을 생성하게 된다. 또한 응용프로그램의 요청이 없을 시기에 선반입을 할 수 있도록 prefetch 요청 모듈을 생성하는 작업을 한다.

Read 요청모듈은 응용 프로그램에서 읽기 요청이 들어올 때 생성된다. 이 모듈은 블록서버와의 통신을 통해 요구하는 블록을 가져올 read 쓰레드를 블록서버

의 개수 만큼 만든다. read 요청모듈은 읽기 요청이 들어오면 먼저 캐시 모듈을 검색하여 요구하는 블록이 로컬 캐시에 있는지 먼저 알아본다. 만약 요구하는 블록이 캐시 내에 없다면 이 모듈은 read 쓰레드를 생성하여 원하는 블록을 가져온다.

Prefetch 요청 모듈은 read 요청모듈과 별도로 생성된다. 이 모듈은 파일서버의 I/O 요청이 없을 때 선반입 쓰레드를 생성하여 블록서버로부터 현재까지 서비스된 블록의 다음 블록을 가져오게 한다.

선반입할 블록의 결정은 병렬 파일 시스템에서 사용되는 파일의 액세스 패턴이 70%이상이 순차적이라는 기존 연구결과를 바탕으로 이루어졌다.

선반입된 블록은 파일서버의 로컬 캐시에 저장된다. 다음 파일서버에서의 읽기 요청 때는 이 블록이 디스크가 아닌 파일서버의 캐시에 있으므로 디스크 I/O 로 인한 액세스 시간을 줄일 수 있게 된다.

3.2 블록서버 선반입 구조

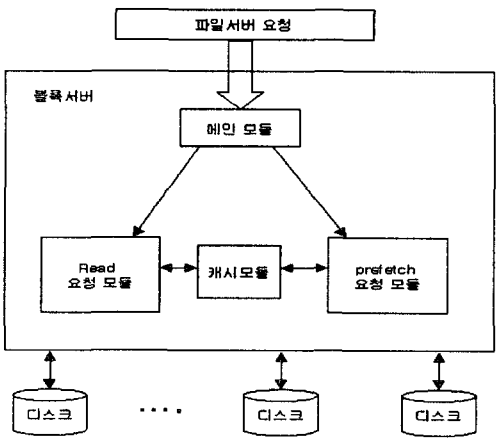

그림 2 블록서버 선반입 구조

블록서버 구조는 파일서버의 구조와 크게 다르지 않다. 기존의 구조와 크게 달라진 점은 블록서버에는 없었던 prefetch 요청 모듈을 만들었다는 것이다<그림 2>.

블록서버의 메인 모듈은 파일서버로부터 오는 요청을 처리할 모듈을 생성하는 작업을 수행한다. 블록 서버의 메인 모듈에서 보면 파일서버의 읽기 요청과 선반입 요청은 같은 요청으로 보게 된다. 이 요청이 들어오면 메인 모듈은 작업을 처리할 블록서버 read 요청모듈을 생성한다. 그리고 이단 선반입 구조를 위해서 블록서버 prefetch 요청 모듈을 별도로 생성한다.

Read 요청 모듈은 파일서버에서 들어온 요청을 처리하기 전에 먼저 로컬 캐시를 검색한다. 이때 서비스할 블록이 캐시 내에 없을 때 read 쓰레드를 이용하여 서비스할 데이터 블록을 디스크로부터 읽어온다.

블록서버의 prefetch 요청 모듈은 파일서버로부터의 읽기 요청과 별도로 요청된 블록의 다음 블록을 디스크로부터 읽어와서 블록서버의 로컬캐시에 갔다 놓는다. 이렇게 가져온 블록은 다음 파일서버의 읽기 요청이든 선반입 요청이 들어올 때 디스크의 액세스 없이 캐시에서 서비스되어진다.

3.3 이단 선반입 액세스 패턴

그림 3 은 이단 선반입 기법의 선반입 액세스 시간과 기존 선반입 기법의 액세스 시간과의 차이를 나타낸 것이다. 액세스 패턴이 A,B 이고 블록 A 의 읽기 요청 뒤에 블록 B 의 선반입 과정을 나타낸다.

우선 그림 (a)는 기존의 선반입 구조로 파일서버의 측면에서 보면 b 에서 블록 A 를 읽은 뒤 블록 서버에 선반입 요청을 하게 된다.그러면 요청을 받은 블록 서버는 B 블록의 선반입을 위해 c 에서 디스크로부터 블록을 읽어온다. 마지막으로 d 에서 B 블록을 파일서버로 보내면서 선반입을 최종 완료하게 된다. 여기서 선반입으로 소요된 시간은 총 (b + c + d)의 시간이 걸리게 된다.

반면에 그림 (b)는 이단 선반입 구조의 선반입 과정을 나타낸다. 구조는 블록 서버에서 a 시간에 블록 A 읽기 요청을 할 때 미리 블록 서버에서는 B 를 읽을 준비를 한다. 그리고 b 에서 블록 A 의 읽기 요청이 끝날 때 블록 서버에서는 디스크에서 벌써 블록 서버 캐시로 다음의 B 블록을 미리 읽어다 놓는다.

그 결과 이단 선반입 기법에서 선반입으로 소요되는 시간은 파일 서버 측면에서 보면 총 (b + c)의 시간으로 일반 선반입구조 보다 선반입으로 소요되는 시간을 줄일 수 있다.

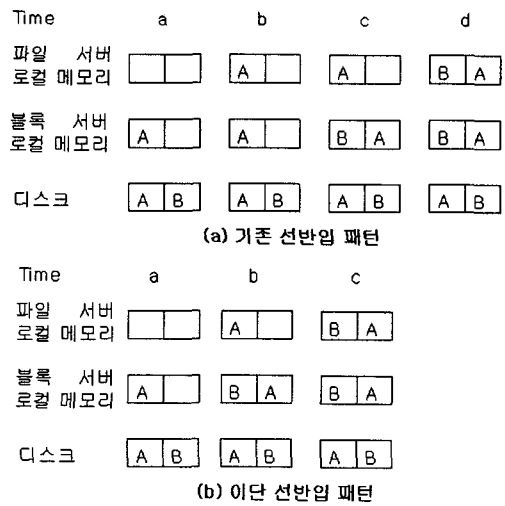

그림 3 블록 액세스 패턴

그림 3 과 같은 패턴을 보면 이단 선반입 기법은 파일서버쪽 선반입 수행과 블록서버쪽 선반입 과정이 비동기적으로 일어남을 알 수 있다.

이는 파일서버 측면에서 볼 때 요청하는 데이터가

디스크가 아니라 블록서버의 로컬 캐시에 있게 되므로 데이터의 액세스 시간을 줄일 수 있다.

4. 실험결과

실험 환경은 리눅스를 기반으로 하는 PFSL 에서 블록서버의 소스를 일부 수정하여 실험을 하였다. 사용되는 응용프로그램은 기존에 연구된 병렬 파일 시스템 액세스 패턴이 거의 순차적이라는 기반으로 순차적인 액세스 패턴을 가지는 응용프로그램을 이용하여 실험하였다.

그림 4 는 순차적 패턴 형식으로 100Mbyte 정도의 파일을 액세스 하여 얻은 10 회의 측정치를 평균한 값이다. 이때 읽어 들이는 블록 단위는 8Kbyte 이며 응용프로그램에서 한번에 읽어 들이는 크기는 512Kbyte 이다. '*Computation 시간 / 프로그램 총 수행 시간*' 의 수치는 적을수록 I/O 위주의 프로그램을 나타내고 있으며, 이는 선반입을 할 충분한 시간이 없음을 나타낸다.

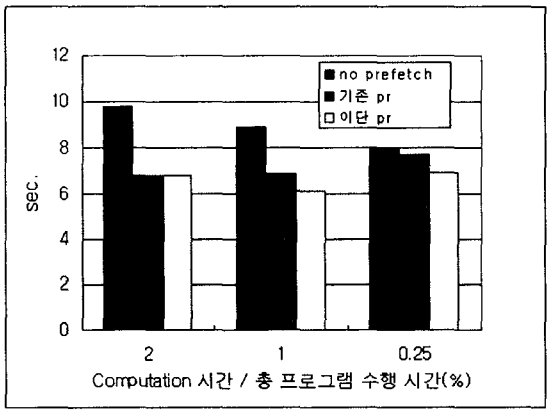

그림 4 각 선반입 기법별 프로그램 총 수행시간

그림 4 에서 나온 결과에서 알 수 있듯이 computation 시간이 길어 선반입할 충분한 시간이 있을 때는 기존 선반입과 이단 선반입의 성능 차이가 없음을 알 수 있다.

파일을 액세스 하는데 있어 I/O 사이의 시간적인 갭이 작아지면 기존의 선반입 기법은 이단 선반입 기법에 비해 큰 성능 향상을 보이지 못함을 알 수 있다. 이는 읽기 요청 뒤에 선반입 액세스가 수행되게 되는데, 이 액세스 시간이 다음 읽기 요청 때 까지 충분하지 않아 다음에 일어나는 읽기 요청과 중첩되어 시스템에서 제공하는 입출력 대역폭을 넘어서기 때문에 성능 향상을 보이지 못하고 있는 것이다.

이런 상황일 때 블록 서버쪽에서 이단 선반입을 한다면 기존의 선반입에 비해 상당한 성능향상을 보인다는 것을 알 수 있다. 이는 블록 서버에서 파일 서버에서 요구하는 데이터를 넘겨준 뒤에 별도로 다음의 예상되는 데이터블록을 블록 서버쪽 로컬 메모리에 읽어다 놓기 때문이다. 그러면 다음 액세스 즉, 선반입 요청 때는 액세스 시간이 줄어들어 다음 읽기 요청에 영향을 주지 않는다. 즉, 파일 서버쪽에서의 선반입 부하를 블록 서버쪽에서 나누어 가지므로 디스크 입출력대역폭을 늘릴 수 있게 되는 것이다.

5. 결론 및 향후과제

점차 사용자측에서 요구하는 파일의 크기가 대용량으로 변해가는 추세에서 싱글 디스크 입출력 속도로는 사용자가 원하는 입출력 대역폭을 제공할 수가 없다.

이 문제를 해결하기위한 방법으로 입출력 부하를 다중서버로 분산시켜 입출력 대역폭을 확장한 병렬 파일 시스템이 있다. 여기에다 선반입 구조까지 적용시키면 더 나은 입출력 대역폭을 얻을 수 있다. 하지만 I/O 위주의 프로그램에서 I/O 사이의 갭이 너무 짧아 선반입할 충분한 시간을 주지 못 한다면 기존의 선반입 구조는 현재 필요한 액세스를 지연시키게 된다. 이런 이유로 일반적인 선반입으로는 뚜렷한 성능 개선을 가지지 않는다는 것을 확인 할 수 있다.

따라서 I/O 위주의 프로그램에서 기존의 선반입 기법 자체가 최선이 될 수 없을 때 이를 개선하기 위한 선반입 구조로 선반입 액세스 자체 부하를 나누어 가지는 파일 서버와 블록 서버의 이단 선반입 구조가 제안되었다. 이 구조는 선반입 액세스 시간을 줄여 프로세서가 연속된 데이터를 요구하는 속도가 빠른 경우에도 큰 성능 향상을 보여준다.

그리고 사용자가 원하는 입출력 대역폭이 선반입으로 인해 시스템에서 제공해주는 입출력 대역폭을 넘어선다면 선반입 자체가 꼭 필요한가를 고려하여 만약 이 선반입 자체가 부하가 된다면 선반입을 중지하는 동적인 선반입 구조가 추가 되어야 할 것이다.

참고문헌

[1] M. K. Tcheun, H. Yoon, and S. R. Maeng, "An Adaptive Sequential Prefetching Scheme in Shared Memory Multiprocessors," in International Conference on Parallel Processing, pp. 306-313, IEEE, Sep. 1997

[2] Jong-Hyun Cho, Chei-Yol Kim, Dae-Wha Seo. "Parallel File System Using Dual Caches Scheme and Prefetching", The 2000 International Conference on Parallel/Distributed Processing Techniques and Application (PDPTA2000), June 2000

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

[4] Toni Cartes, "Cooperative Caching and Prefetching in Parallel/Distributed File System", Ph. D Thesis, Universitat Politecnica de Catalunya, 1997

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