2000년 한국정보처리학회 추계 학술발표논문집 제 7 권 제 2 호

파일 액세스 패턴과 캐쉬 영역을 고려한 선반입 기법

임재덕, 조종현, 서대화
경북대학교 전자공학과
e-mail : jdscol92@palgong.knu.ac.kr, alfcom@palgong.knu.ac.kr, dwseo@ee.knu.ac.kr

Prefetching Policy based on File Access Pattern And Cache Size

Jae-Deok Lim, Jong-Hyun Cho, Dae-Wha Seo
Dept. of Electronic Engineering, Kyungpook National University

요 약

선반입 및 캐싱에 관한 기법은 병렬 파일 시스템 등의 저장 장치의 입출력 성능을 개선하는 효과적인 방법으로써 현재까지 여러 가지 연구가 많이 이루어져 왔다. 특히 선반입 기법은 응용프로그램에서 사용할 데이터 블록을 디스크로부터 미리 가져 옴으로써 실제 필요할 때의 디스크 입출력 횟수를 줄여 실행 시간을 단축시키는 방법이다. 본 논문에서는 리눅스 기반의 병렬 파일 시스템 상에서 선반입 되는 블록의 수를 제한하여 캐쉬의 효율성을 높이고 실행 시간을 단축시키는 선반입 기법을 구현하였다. 특히 순차적인 접근을 하는 대용량 파일에서 그 성능이 개선됨을 보여 준다.

1. 서론

최근 컴퓨터 시스템의 발전 경향을 보면 디스크와 같은 저장 장치의 성능은 CPU 등과 같은 컴퓨팅 장치의 성능에 비해 괄목할 만한 발전을 이루지 못했다.

따라서 이런 성능차이를 줄이기 위한 연구가 많이 이루어졌다. 캐싱은 사용되는 파일의 지역성이 클 경우 입출력 횟수를 줄임으로써 파일 블록들의 빠른 접근을 가능하게 한다. 하지만 사용되는 파일의 크기가 점점 커지고 한 번만 읽혀지는 블록들의 증가로 캐쉬의 효율성 문제가 발생한다[1]. 이런 한계를 보완하기 위해 앞으로 사용될 블록들을 미리 가져오는 선반입 기법은 디스크에 대한 접근 시간을 줄이는 좋은 방법이 된다[1,6,7].

하지만 무조건적인 선반입은 파일의 액세스 패턴이 순차적이지 않을수록 오히려 캐쉬의 효율성을 떨어뜨릴 수가 있다. 파일의 액세스 패턴이 순차적이지 않을 경우 선반입 된 블록들이 앞으로 사용될 확률은 적어지며, 캐쉬 내에 존재하는 앞으로 사용될 블록들을 대체할 가능성이 커진다. 따라서 캐쉬 내의 블록과 선반입 된 블록들이 각각 관리되기 보다는 캐쉬 영역으로 선반입되는 블록들의 양을 조절할 수 있는 통합적인 관리가 필요하다[2].

본 논문에서는 캐쉬영역 내에서 선반입 영역의 비율을 제한하여 데이터 블록들을 선반입하는 기법 즉, 캐쉬영역 내에 선반입 될 수 있는 블록들의 최대 개수를 제한하여 선반입하는 기법을 구현하였다. 적용된 선반입 기법은 OBA(One-Block Ahead) 기법을 확장한 E-OBA(Extended One-Block Ahead) 기법을 적용한다. 일반적인 병렬 파일 시스템 상에서 자주 나타나는 몇 가지 패턴들을 구현된 기법에 적용하여 시스템의 성능을 비교 분석하였고, 그 결과를 각 패턴에 따라 동적으로 적용하여 시스템이 최적의 성능을 낼 수 있도록 하였다.

2. 선반입 기법

단순히 현재 블록의 다음을 가져오는 방법 이외에도 좀 더 정확한 블록들을 예측하여 선반입하는 기법들이 이전에 많이 연구되었다.

현재 읽혀진 블록 다음에 올 블록들의 형태를 가장 잘 알고 있는 것은 그 블록들을 사용하는 응용 프로그램이라는 것에 착안하여 응용 프로그램이 선반입할 블록에 대한 정보를 시스템에 알려주는 Transparent Informed Prefetching 기법이 Scotch file System 에서 제안되었다[1].

데이터 압축에 사용되는 Lempel-Ziv 알고리즘이 선반입 기법에 사용되었다. 이 기법에서는 확률 트리의 데이터 구조를 생성, 유지하여 앞으로 사용될 블록들을 예측할 수가 있는데 시스템이 작동하면 할수록 확률 트리가 수정이 되고 그 크기도 점점 커지게 된다[6].

데이터 압축을 기반으로 구현된 알고리즘을 간소화 한

ISG (Interval-and-Size Graph)라는 알고리즘에서는 응용 프로그램이 정규적인 형태로 파일을 액세스 한다는 사실을 바탕으로 한다. 따라서 일단 파일이 한 번 액세스 되면 그 형태가 액세스 되는 블록들 간의 간격 및 크기 등의 정보를 기반으로 그래프 형태의 자료 구조로 나타나게 되며 이를 참조하여 앞으로 사용될 블록들을 예측하게 되는 것이다[7].

위의 방법들은 앞으로 사용될 블록들을 예측하여 미리 가져옴으로써 디스크 입출력의 시간을 줄이는데 효과적인 성능을 나타낸다. 하지만 사용할 블록을 예측하기 위한 복잡한 자료 구조 등을 유지하고 매번 갱신해야 하며, 사용할 파일의 액세스 패턴을 얻어내기 위해 한번은 파일의 액세스가 이루어져야 하는 단점이 있다. 또 병렬 및 분산 시스템에 사용되는 파일의 액세스 패턴을 보면 대부분이 순차적이며 그 크기도 대용량화 되고 있다[3,4]. 이런 이유로 본 논문에서는 시스템의 선반입 기법으로 OBA 기법을 확장한 E-OBA 기법을 제안하고 시스템에 적용하였다.

3. E-OBA 선반입 시스템 구조

3.1. PFSL 시스템 구조

본 논문에서는 PFSL(Parallel File System for Linux) 시스템을 기반으로 구현되었고[5], 시스템 구조는 그림 1 과 같이 크게 파일서버, 블록 서버 그리고 메타데이터 서버로 구성되어 있다.

파일서버는 실행 노드에 위치하여 파일 수준의 처리에 대한 응용프로그램 인터페이스를 제공한다. 블록서버는 I/O 노드에서 파일의 데이터 블록들을 관리 및 서비스한다. 메타데이터 서버는 병렬 저장된 파일의 메타 데이터를 관리하며 블록 서버가 정상적으로 동작하는지를 주기적으로 검사한다.

응용프로그램이 파일서버로 서비스를 요청할 경우 시스템은 동작한다. 이 때 파일서버는 요청된 파일의 저장 정보 등을 메타데이터 서버로부터 얻어내어 요청된 파일의 데이터 블록에 대한 서비스를 블록 서버에게 요청한다. 블록 서버는 요청된 블록을 파일 서버로 서비스해 준다. 디스크를 통한 데이터 블록의 요청이 없을 경우 파일서버에서는 선반입이 이루어진다.

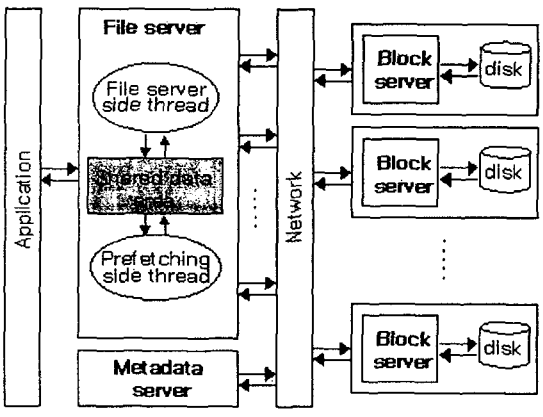

그림 1. 선반입이 구현된 PFSL 구성도

3.2. 파일 서버

그림 2 는 파일서버 내에 구현된 선반입 모듈의 상세구조이다. 파일 서버는 크게 응용프로그램의 요청을 서비스하는 파일서버 쪽의 수행영역과 응용프로그램의 요청과 관계없이 선반입을 수행하는 영역으로 나뉘어진다.

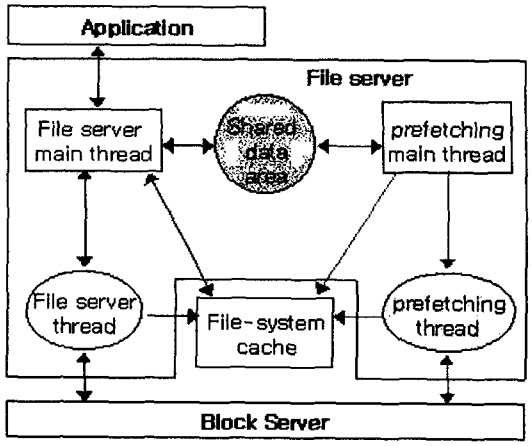

그림 2. 파일 서버 내의 선반입 모듈 구조

각각은 스레드로 동작되며 비동기적으로 수행되지만 공유 데이터 영역을 통해 무분별한 선반입을 막고 제한된 조건 내에서 선반입 할 수 있도록 한다. 파일서버 메인 스레드는 응용프로그램으로부터 요청이 있을 경우 서비스할 데이터 블록을 블록서버에게 요청하기 전에 디스크의 입출력 횟수를 줄이기 위하여 캐쉬를 참조한다. 이 때 캐쉬를 참조하여 원하는 데이터 블록이 있을 경우 디스크 입출력 횟수를 줄일 수 있다. 만약 원하는 데이터 블록이 캐쉬 내에 없을 경우 파일서버 스레드들을 생성하여 블록 서버들에게 데이터 블록을 요청한다. 이 경우 파일 서버 쪽은 현재 파일의 접근 위치를 공유 데이터 영역에 저장하고 이 위치를 참조하라는 표시를 하여 선반입시 참조하도록 한다.

3.3. 공유 데이터 영역

선반입 루틴은 수행 시 올바른 블록을 그리고 제한된 조건 내에서 선반입이 이루어지도록 본 논문에서 구현된 공유 데이터 영역을 참조한다. 공유 데이터 영역은 파일서버 루틴과 선반입 루틴이 공유하는 변수들로 이루어지며 다음과 같은 변수들로 구성된다.

응용프로그램이 파일서버에게 데이터를 요청하는 위치를 저장하는 위치 구조체, 선반입 시에 이 위치 구조체를 참조할 것인지 혹은 선반입 된 위치를 기준으로 선반입을 할 것인지를 결정하는 위치 참조 변수, 현재까지 선반입 된 블록의 개수, 선반입 영역으로 할당된 영역이 선반입 된 블록으로 가득 찼는지의 여부를 나타내는 선반입 영역 확인 변수 등이 있다. 파일 서버는 응용프로그램의 요청을 서비스 하면서 위치 구조체 및 위치 참조 변수 등을 갱신한다. 선반입 루틴은 선반입을 수행하면서 위의 변수를 참조하며 선반입 된 블록의 개수 등을 갱신한다.

3.3. E-OBA (Extended One Block Ahead) 기법

일반적인 OBA 기법은 응용프로그램이 파일에 대한 서비스를 요청할 경우 요청한 블록 다음 것도 같이 서비스하는 방식이다. 하지만 응용프로그램이 파일 블록에 대한 서비스 요청을 하는 경우에 한해 다음 블록이 선반입되므로, 응용프로그램의 데이터 블록에 대한 요청들 사이의 시간이 지연된다면, 입출력 대역폭에 여유가 있다 하더라도 선반입을 수행하지 않아 결과적으로 선반입의 효과가 떨어진다.

따라서 본 논문에서는 이런 문제를 감소하기 위해 OBA

기법을 확장한 E-OBA 기법을 사용한다. E-OBA 기법은 응용 프로그램의 요청과 관계없이 제한된 블록 개수만큼 계속해서 선반입을 하는 방식이다.

선반입 위치는 공유 데이터 영역의 위치 참조 변수에 따라 달라진다. 응용프로그램으로부터 파일 블록에 대한 요청이 있을 경우 파일서버는 응용프로그램의 요청이 끝나는 파일의 위치를 공유 데이터 영역의 위치 구조체에 기록하고 선반입 시에 이 위치 구조체를 참조하도록 위치 참조 변수에 표시한다. 선반입 루틴은 선반입 전에 공유 데이터 영역의 위치 참조 변수를 검사한다. 참조하라는 표시가 있으면 저장된 위치 구조체의 파일 위치를 기준으로 선반입을 하고 위치 참조 변수에 참조했다는 표시를 한다. 참조하라는 표시가 없을 경우와 참조 했다는 표시일 경우에는 현재까지 선반입한 위치를 기준으로 제한된 블록 개수만큼 선반입을 진행한다.

3.4. E-OBA 선반입 절차

선반입 메인 스레드는 선반입 수행을 관리하는 스레드로써 응용프로그램이 파일에 접근하기 위해 파일 열기 요청을 할 경우 생성되며 파일이 닫히면 종료된다. 선반입 메인 스레드는 선반입을 시작하기 전에 항상 공유 데이터 영역을 참조하여 선반입 조건을 검사한다.

선반입 조건은 선반입 시작 위치가 파일의 끝을 넘어 서는지, 선반입 된 블록의 개수가 제한된 개수를 초과했는지 등의 여부를 조사하는 것이다. 이 조건이 만족되면 공유 데이터 영역으로부터 선반입을 시작할 파일의 위치를 얻어내어 그 위치로부터 선반입을 시작한다.

선반입 후 선반입된 블록의 개수 및 선반입한 위치 등의 변수를 공유 데이터 영역에 갱신한다. 선반입 스레드는 블록서버로부터 데이터를 선반입하기 위해 블록서버 개수만큼 생성되고, 선반입이 완료되면 선반입 스레드는 종료된다.

선반입 수행은 스레드를 이용하여 구현되었으므로 비동기적인 입출력이 가능해져 시스템의 성능을 개선할 수 있다. 응용 프로그램이 원하는 데이터 블록을 선반입하기 위해서는 선반입을 수행하는 위치가 중요한 변수로 작용한다. 또한 선반입 되는 블록의 수가 많아질수록 캐쉬 내에 존재하는 블록들이 다음에 사용될 블록들일지라도 대체될 가능성이 커진다. 이런 이유로 선반입되는 블록들의 위치 및 그 양의 조절이 중요해 진다[2]. 따라서 공유 데이터 영역을 참조한 E-OBA 선반입 기법은 선반입 블록의 위치 및 선반입 블록의 개수 문제를 완화할 수 있다.

4. 실험 및 결과

실험은 수정된 PFSL 시스템 상에서 하였고[5], 사용된 응용 프로그램은 기존에 연구되었던 액세스 패턴을 기반으로 하여 그와 유사한 패턴으로 접근이 되도록 하였다[3,4].

4.1. 사용된 패턴 형태

그림 3 은 실험에 사용된 액세스 패턴이다.

(a)와 같은 패턴은 일반적인 파일 시스템을 포함하여 대부분의 응용프로그램에서 발견되는 형태이고, (b)와 같은 패턴은 병렬 처리 시스템 상의 과학 기술 계산 응용에서 주로 나타나는 패턴이며 (c)와 같은 패턴은 그 이외의 나머지 모든 패턴을 포함한다.

4.2. 실험 결과

그림 4 는 위에 정의된 패턴형태로 100Mbytes 정도의 한 파일을 액세스하여, 각각 4 회의 실험 결과를 평균한 것이다. (a)는 100Mbytes 정도의 파일을 전체적으로 한 번 읽은 결과이고, (b)(c)는 주어진 액세스 패턴으로 파일을 읽은 전체 크기가 100Mbytes 정도일 때의 결과이다.

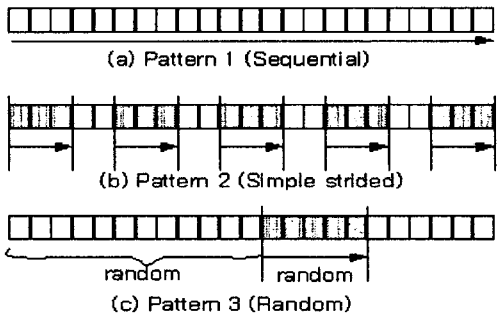

그림 3. 사용된 액세스 패턴

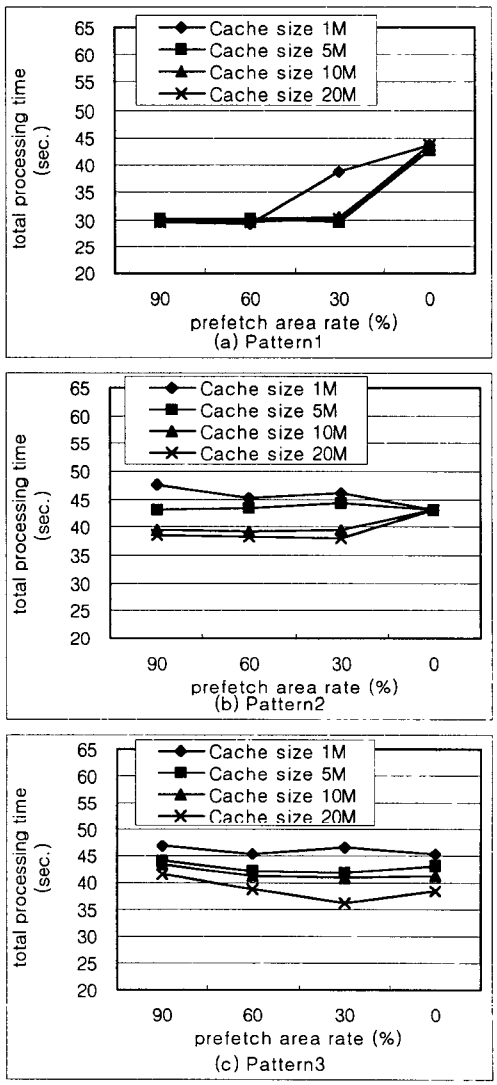

그림 4. 각 패턴에 따른 수행 시간

(a)의 결과에서 알 수 있듯이 순차적인 패턴인 경우 캐쉬의 크기가 파일의 크기와 같거나 크지 않은 이상 선반입이 없을 경우 그 성능은 거의 차이가 없지만, 선반입이 이루어지면 매우 좋은 성능을 낼 수 있다는 것을 알 수가 있다. 순차적인 접근형태를 가지는 파일의 특성상 한 번 읽혀진 것은 가까운 시기에 다시 읽혀질 가능성이 거의 없다는 특성 때문이다.

(b)의 패턴모양은 줄무늬 모양이지만 논리적으로는 순차적 패턴과 비슷하기 때문에 선반입의 영향을 받는다. 하지만 최악의 경우 선반입 된 블록이 잘못 예측될 가능성이 조각난 파일의 끝에서 마다 일어날 수 있으므로 이중적인 입출력의 가능성이 있게 된다. 결과적으로 순차적인 접근 형태와 비슷하게 캐쉬가 커질수록 선반입 영역에 크게 영향을 받지는 않지만 대략 30~60% 정도의 영역에서 성능이 좋으며, 캐쉬 크기가 적을수록 선반입 수행이 오히려 과부하가 될 수 있다.

(c) 와 같은 패턴은 수행시기마다 읽혀질 블록의 위치, 연속적으로 읽혀질 블록의 수가 다르기 때문에 어떤 정형화된 결과를 얻을 수는 없다. 하지만 캐쉬의 크기가 적을 때의 선반입에 의한 블록은 오히려 캐쉬 내의 앞으로 읽혀질 블록들의 대치 가능성이 커지므로 과부하로 작용할 가능성이 크고, 캐쉬크기가 커질수록 선반입 수행이 없어도 수행시간이 단축됨을 알 수가 있다. 하지만 캐쉬가 커질수록 어느 정도의 선반입이 성능 개선에 도움이 됨을 보여 준다.

5. 결 론

병렬 파일 시스템에서 사용되는 패턴의 대부분이 순차적이라는 결과로 본다면 선반입 영역을 캐쉬 영역에 대해 크게 잡을수록 시스템의 성능을 약 20~30% 가량 개선함을 알 수 있다. 순차적이지 않은 패턴들에 대해서는 캐쉬 크기가 작을 경우 오히려 선반입에 의한 과부하가 생길 수도 있지만, 캐쉬크기가 큰 경우 캐쉬 영역에 대해 약 30%~50% 정도의 선반입 영역을 가지고 선반입을 할 경우 약 3~5% 정도의 시스템 성능이 개선됨을 알 수 있다.

따라서 큰 데이터를 다루는 병렬 파일 시스템이라든지 멀티 프로세싱에서는 선반입 수행이 필수적으로 행해져야 한다. 나아가서는 분석된 결과를 기반으로 시스템 자체에서 파일의 액세스 패턴을 분석하여 파일의 액세스 패턴에 맞는 정책을 결정하도록 하는 동적인 기법이 추가되어야 한다.

참고문헌

[1] R, Hogo Pattern and Garth A. Gibson. "Exposing I/O concurrency with informed prefetching". In Proc. Third International Conf. on Parallel and Distributed Information Systems, September 1994

[2] Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li. "A study of integrated prefetching and caching strategies". In Proc. 1995 ACM SIGMETRICS, pages 188-197, May 1995

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

[4] 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, pp. 169-180, Vol. 1245, June 1997

[5] 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

[6] Curewitz, K. M., Kristian, P., And Vitter, J.S. "Practical prefetching via data compression". In Proceedings of the SIGMOD Management of Data, Pages 257-266. ACM Press, May 1993

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