2001년 한국정보처리학회 춘계 학술발표논문집 제8권 제1호

병렬 파일 시스템에 적용한 테이블 비교 선반입 기법

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

Table Comparison Prefetching Algorithm Adoped in Parallel File System

Yoon-Young Lee*, Chei-Yol Kim**, Dae-Wha Seo*
*Dept of Electronic Engineering, Kyungpook National University
**Linux Research Team, ETRI

요 약

과도한 파일 입출력이 요구되는 병렬파일 시스템의 성능을 결정하는 중요한 요소로서 캐쉬과 선반입을 들 수 있다. 본 논문은 캐쉬의 크기에 비해 상대적으로 큰 파일을 요청하는 경우에 시스템 성능에 막대한 영향을 미치는 선반입에 대해서 선반입할 데이터를 결정하는 알고리즘으로 테이블 비교법을 제안하고, 이와 더불어 예측된 데이터의 선반입 여부와 선반입 시기를 결정하는 경우 현재의 가용 입출력 대역폭을 고려하는 기법을 제안한다. 제안하는 선반입 알고리즘은 실제 병렬파일시스템에 구현되었으며, 파일시스템 성능향상에 크게 기여하였다.

1. 서론

프로세서와 네트웍 장비의 급속히 발전한데 반하여, 입출력 장치의 발전 속도는 이를 따라잡지 못하고 있다. 이 때문에 입출력 장치가 전체 컴퓨터 시스템의 병목이 되어 버렸다. 이러한 병목을 해소하는 하나의 방법으로 입출력 장치에 병렬성을 도입하였으며 이 결과로 현재 많은 클러스터링 컴퓨팅 환경에서 병렬 입출력 기법(Parallel I/O)과 병렬 파일 시스템(Parallel File System)이 사용되고 있다. 병렬 파일 시스템의 도입으로 파일의 입출력 대역폭은 향상이 되었으나 여전히 파일 요청시의 디스크 접근 지연시간을 줄일 수는 없었다. 이러한 문제의 해결책으로 이전에 사용한 데이터 블록 중 다음에 사용될 확률이 높다고 생각되는 것들을 메모리에 남겨두는 캐쉬과 다음에 사용될 것으로 예측되는 블록을 미리 메모리로 읽어 들여 파일 서비스 요청 시 지연시간을 최소화하는 선반입 기법이 있다. 그런데, 과학 계산용 병렬 응용프로그램에 사용되는 파일의 경우에는 대부분의 파일 크기가 버퍼 캐쉬의 크기보다 크며 요구하는 데이터의 양도 일반적인 파일 시스템에 비해 대부분 크다[4][5]. 이러한 경우 캐쉬보다 선반입 기법이 파일 시스템 성능에 훨씬 많이 기여하게 된다.

선반입 기법에 대한 이전의 연구는 대부분 다음에 선반입할 대상이 되는 블록을 예측하는 알고리즘이 대부분이었다. 하지만 실제 파일 시스템 성능 면에서 본다면 파일 요청이 과도한 경우에는 선반입을 하지 않는 것이 성능에 유리한 경우도 발생한다. 이를 고려해 본 논문에서는 선반입할 데이터를 결정하는 알고리즘으로 두 개의 테이블을 비교해서 패턴을 찾아내는 테이블 비교 기법을 소개하고, 결정된 데이터의 선반입 여부와 선반입 시기를 결정하는데 있어 응용프로그램들의 각 파일에 대한 단위시간당 데이터 요구량을 산출하여, 현재 사용 가능한 입출력 대역폭을 계산하고, 이를 기준으로 선반입 시기를 결정하는 가용 입출력 대역폭을 고려한 선반입 정책을 소개한다. 그리고 이를 실제로 병렬 파일시스템에 구현하여 그 성능을 평가한다.

2. 관련 연구

2.1. 병렬 응용프로그램의 파일 접근 패턴

최근에 와서야 진행된 클러스터링 환경에서의 병렬 응용프로그램의 파일 접근 패턴 연구[4][5]에 따르면 대부분의 접근 패턴은 크게 3가지로 나누어진다. 첫째, 순차적 접근(sequential)으로 하나의 계산 노드가 자신의 개인 파일을 순차적으로 접근하는 경우이며, 둘째로는 하나의 파일을 여러 노드가 동시에 접근하여 자신이 맡은 해당 부분을 읽어 가는 경우(interleaved)이며, 마지막으로 위의 두 가지 경우를 혼합한 경우(mixed)로 파일의 헤더 부분은 하나의 노드가 순차적으로 접근하고 나머지 부분은 두 번째 접근 패턴으로 여러 노드가 동시에 접근하는 형태이다.

2.2. 병렬파일 시스템에서의 선반입 기법

이상적인 경우의 선반입 알고리즘은 응용프로그램이 다음에 요구할 데이터를 정확히 알고 선반입하는 경우이다. 이러한 접근 방향의 연구로서 응용프로그램이 제공하는 힌트를 사용해서 선반입하는 방식이 있다[1]. 그러나 힌트를 사용하는 방식은 응용 프로그래머의 부가적인 작업이 필요하고, 기존 응용프로그램에 대한 호환성이 없다는 이유로 널리 사용되지 않고 있다. 이와는 달리 응용프로그램이 제공하는 파일 접근 패턴에 대한 정보 없이 현재의 파일 접근 기록이나 과거의 파일 접근 기록을 가지고 다음에 사용될 데이터를 예측하는 방식의 연구가 있다. 이러한 연구로는 Griffioen과 Appleton[2]이 제안한 확률 그래프를 사용하는 방법이나 Cortes가 제안한 ISG(Interval-and-Size-Graph)방법[3]등이 있다. 확률 그래프는 파일 크기가

대부분 매우 작은 일반 파일 시스템에서 선반입 단위를 파일로 보고 만든 것으로써 병렬파일 시스템에 적용하기는 적절하지 않다. ISG방식은 기타 다른 방식에 비해 유지해야 하는 자료의 양을 줄였음에도 불구하고, 여전히 패턴이 일정하지 않은 경우에는 그래프를 유지하는 비용이 과도하게 커질 수 있어 시스템의 과부하로 작용할 여지가 있다.

3. PFSL(Parallel File System for Linux)

본 논문은 우리가 개발한 PFSL[6]을 기반으로 하고있다. PFSL은 크게 클러스터 파일 매니저, 블록 매니저 그리고 메타데이터 매니저로 나뉘어지며, 그림 1과 같이 구성된다. 클러스터 파일 매니저는 실행 노드에서 응용프로그램이 요청하는 파일 서비스를 담당하고, 블록 매니저는 입출력 노드에서 분산 저장된 파일에 대한 데이터 블록의 입출력 서비스를 담당한다. 메타데이터 매니저는 분산 저장된 파일에 대한 메타데이터를 관리하는 서버이다.

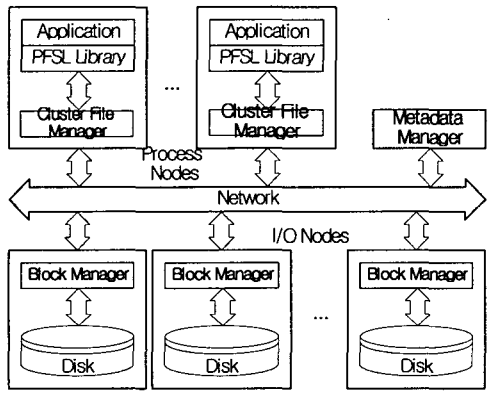


그림 1 PFSL의 구성

응용프로그램이 PFSL 라이브러리를 통해 클러스터 파일 매니저에게 파일에 대한 서비스를 요청하면, 클러스터 파일 매니저는 응용프로그램의 요구를 수행하기 위해 메타데이터 매니저에 있는 메타데이터를 이용하여 필요한 데이터 블록의 논리적 주소를 연산한 후, 입출력 노드의 블록 매니저에게 데이터 블록에 대한 서비스를 요구한다. 블록 매니저는 클러스터 파일 매니저로부터 데이터 블록에 대한 서비스 요청이 있을 경우, 디스크의 입출력을 통해 데이터 블록의 읽기/쓰기 요청을 서비스한다.

클러스터 파일 매니저에서 멀티쓰레드를 이용해서 동시에 여러 블록 매니저로 데이터 블록을 요구하여, 블록 매니저에서의 데이터 블록 서비스가 병렬로 진행될 수 있다. 이러한 병렬 처리의 장점은 입출력 노드가 많아질수록 데이터의 입출력에 대한 성능 향상을 가져온다.

4. 테이블 비교 선반입 기법

테이블 비교 선반입 기법은 응용프로그램으로부터 파일 요청이 들어오면 각 파일에 대하여 그림1에 보이는 테이블 A,B 두 개를 각각 생성한다. 테이블 A,B는 요청이 들어 올 때마다 번갈아 갱신된다. request_size는 요구한 블록의 개수를 나타내며, requets_interval은 이전 요청의 마지막 블록과 현 요청의 첫번째 블록간의 간격을 기록한다. start_block_num은 요청한 블록들의 첫번째 블록 번호를 나타내며 이는 다음 요청 때의 request_interval을 계산할 때 사용된다. 즉, request_interval_A(B) = start_block_num_A(B) - (start_block_num_B(A) + request_size_B(A) - 1) 이다.

이렇게 기록된 두개의 테이블을 비교하여 request_size와 request_interval이 일치하는 경우에 일정한 패턴이라고 판단하고 다음에 선반입할 데이터 블록을 결정한다.

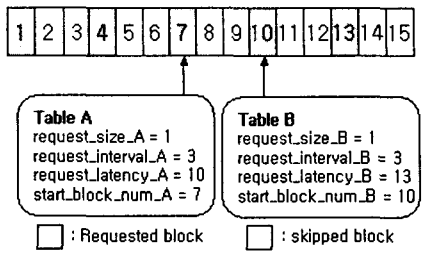


그림 2 simple strided 패턴에 대한 테이블 비교 선반입의 예

그림2는 파일 요청이 1,4,7, · · · · 의 순으로 블록을 요청하는 simple strided 패턴에 대한 테이블 비교법의 예를 들고 있다. 그림1에서의 판단 결과는 Table A와 Table B의 request_size와 request_interval이 동일하므로 13번을 10번 다음에 요청될 블록으로 결정한다. 특별한 경우로 request_interval이 1인 경우는 순차적 요청을 나타낸다.

기존의 선반입 알고리즘[2][3]은 블록 예측을 위하여 매우 크고 복잡한 자료구조를 사용하고 있다. 그러나 병렬 응용프로그램의 파일 접근 패턴[4][5]을 분석해 보았을 때 2.1절에서 언급한 3가지 형태의 파일 접근이 주류를 이루고 있으며, 이외의 패턴은 전체 요구의 1% 이하이다. 이러한 이유로 제시한 테이블 비교법은 간단한 자료구조만으로도 대부분의 병렬 응용프로그램의 파일 접근 패턴에 대해서 충분한 성능을 발휘한다

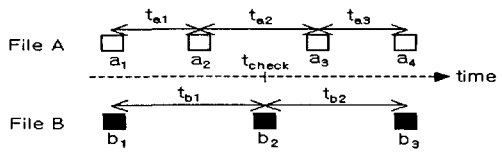
5. 가용 대역폭을 고려한 선반입 여부 및 시기 결정

병렬 응용프로그램에서 사용하는 파일은 대부분이 크기가 커서 응용프로그램은 한 번에 하나의 파일 전체를 요구하지는 않는다. 응용프로그램은 메모리에 일정한 크기의 버퍼를 할당 받고 이를 통하여 파일 데이터를 반복적으로 요청하여 연산작업을 수행한다. 파일 전체에 대해서 이렇듯 동일한 크기의 데이터를 요청하지는 않으나 부분적으로는 이러한 경향을 따른다. 본 논문에서는 이러한 응용프로그램의 파일 접근 경향을 이용해 대강의 사용 가능한 입출력 대역폭을 계산하여 선반입 여부와 시기를 결정하는데 사용한다.

그림3은 응용프로그램이 두개의 파일 A, B를 요청하는 경우를 나타낸다. 시스템의 가용 대역폭을 계산하기 위해서는 먼저 현재 응용프로그램이 사용하고 있는 입출력 대역폭을 계산하는 과정이 필요하다. 그림2의 요구 지연시간(request latency)은 그림1의 테이블에 정의된 request_latency 항목과 같은 것으로 각 요청이 들어올 때마다 테이블에 기록된다.

그림3의 블록 데이터 b2가 요청되었을 때의 시간 t_{check}에서의 가용 입출력 대역폭을 계산해 보자. 먼저 응용프로그램들이 현재 사용하고 있는 입출력 대역폭은 가장 최근의 지연시간과 요구 데이터 크기를 이용해서 구해진다. 따라서, BW_{used} = a2/ta1 + b2/tb1 가 되며 파일 시스템의 최대 대역폭을 BW_{max} 라고 하면 가용 대역폭 BW_{avail}는 아래와 같다

$$BW_{avail} = BW_{max} - BW_{used}$$



• a_n, b_n : request size
• t_{an}, t_{bn} : request latency (n=1,2,3, ···)

그림 3 입출력 대역폭 계산을 위한 파일 요청의 예

선반입의 시기를 조정하기 위한 방법으로 PFSL은 클러스터 파일 매니저 내의 선반입 관리자가 입출력 관리자에게 요청을 전달해 주는 경로를 두 가지로 나누는 방법을 사용하였다.

선반입 관리자(Prefetching manager)는 응용프로그램으로부터 온 파일요청을 기반으로 하여 선반입 정책에 따라 선반입할 데이터를 결정하게 된다. 이때 선반입이 적용되는 요청은 읽기 요청뿐이며 나머지 요청은 그대로 파일요청 큐를 통해 입출력 관리자로 전달된다. 일반적으로 대부분의 파일 시스템은 선반입 관리자 다음에 큐를 하나만 가지고 있다. 이는 응용프로그램의 파일요청과 선반입 요청을 함께 수행하기 때문이다. 이렇게 하면 선반입 관리자의 구조는 상당히 간단해 질 수 있으나 뒤에서 기다리고 있는 파일요청은 선반입 요청 때문에 서비스가 지연 될 수도 있다. 이러한 단점을 피하고자 제안하는 선반입 관리자는 아래의 그림 4와 같이 입출력 관리자에게 요청을 전달하는 두 개의 큐를 지정하는 방법을 사용하였다.

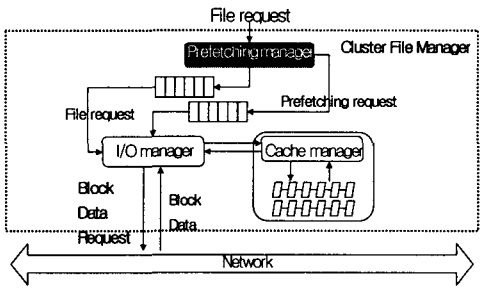


그림 4 선반입 시기결정을 위한
클러스터 파일 매니저 구조

두 개의 큐는 각각 응용프로그램으로부터 온 파일요청을 담당하는 파일요청 큐와 선반입 관리자가 생성한 선반입 요청을 담당하는 선반입 요청 큐이다. 두 개의 큐는 서로 다른 우선순위를 가지고 있다. 즉 선반입 요청 큐에 비해 파일요청 큐가 높은 우선순위를 가지고 있어 큐의 하부에 위치하는 입출력 관리자(I/O manager)는 파일요청 큐에 쌓인 작업이 없는 경우에만 선반입 요청 큐의 작업을 처리하게 된다. 그러나 선반입 요청이라고 해서 항상 선반입 요청 큐에만 들어가는 것은 아니다. 즉 선반입 관리자의 결정에 따라서 어떠한 경우에는 선반입 요청이 파일요청과 함께 파일요청 큐에 들어가기도 한다. 이는 전적으로 선반입 관리자의 정책에 따른 것이다. 이처럼 선반입 관리자 하부에 두 개의 큐를 둠으로써 파일서버는 선반입 당시의 상황에 따라 선반입 시기를 결정할 수 있다.

다음은 가용 대역폭 BW_{avail}을 이용하여 위에서 설명한 시스템에서 선반입 여부와 시기를 결정하는 알고리즘을 나타낸 것이다.

① if ($BW_{avail} < 0$)
  No prefetching
② else if ($BW_{avail} < 1/2\ BW_{max}$)
  if (request_interval == 1)
    Prefetching with current request
  else
    Push prefetching request into prefetching queue
③ else if ($BW_{avail} > 1/2\ BW_{max}$)
    Push prefetching request into prefetching queue

①의 경우는 현재 응용프로그램이 요구하는 입출력 대역폭이 시스템이 제공할 수 있는 능력을 넘어서는 것으로 판단하고 선반입을 중지하게 된다. 이런 경우의 선반입은 당장 필요한 데이터 블록의 입출력마저 방해해 성능을 저하시키기 때문이다.

②의 경우에는 request_interval = 1 이면 파일 요청이 연속적이라는 의미며, 이때는 연속적으로 현재 요청한 블록과 선반입할 블록을 동시에 읽어 들인다. 이렇게 하는 이유는 충분한 입출력 대역폭이 남아 있지 않은 상황에서는 가능하면 파일 시스템의 성능을 효과적으로 이용하기 위하여 연속된 블록은 한 번에 읽어온다. 이렇게 함으로써 뒤에서 기다리는 파일 요청 처리가 조금 더 지연될 수 있지만 전체 파일 서비스 시간을 감안한다면 성능에 도움이 되기 때문이다. 즉 선반입을 하는데 있어 약간의 시간 지연만으로 다음에 들어올 요청을 처리할 수 있는 경우에는 현재의 파일 요청과 더불어 선반입을 하는 것이다. 그러나 request_interval ≠ 1 경우와 같이 선반입을 하기 위한 비용이 일반적인 파일 요청을 처리하는데 드는 비용과 비슷할 경우에는 바로 선반입을 하지 않고 선반입 요청을 선반입 큐에 넣어 입출력 요청이 없을 때 선반입을 수행하게 된다.

③의 경우에는 입출력 대역폭이 충분히 남아있는 상황으로 판단하고 모든 선반입을 입출력 요청이 없을 때 수행하도록 선반입 큐에 넣어 둔다.

위와 같은 알고리즘으로 본 논문에서 현재의 파일 시스템 상황을 고려하여 선반입 여부 및 시기를 결정하게 된다.

6. 실험

6.1 실험 환경

4노드의 테스트 베드를 구성하여 실험하였다. 각 노드는 펜티엄III 600MHz의 CPU를 사용하고 있으며, 128MB의 메모리를 가진다. 각 노드는 Intel의 100Mbps Ethernet 카드와 100Mbps Hub를 이용하여 서로 연결하였다. 다른 네트웍 트래픽의 영향을 받지 않도록 하기 위하여, 테스트 베드만을 허브에 연결하여 독립적인 망을 구성하여 실험하였다. 실험에 사용된 운영체제는 리눅스 커널 2.2.12이다.

6.2 작업 부하

각 작업 부하들은 주로 sequential, interleaved, mixed의 3가지 파일 접근 패턴을 가지고 있다. 작업 부하를 생성할 때, 이 세 가지 패턴을 주로 하였으며, 나머지는 임의의 패턴을 형성하도록 하였다. 실험에서는 데이터 요구량(request rate)를 기준으로 LIGHT, MEDIUM, HEAVY의 세 가지 작업 부하를 생성하여 실험하였다.

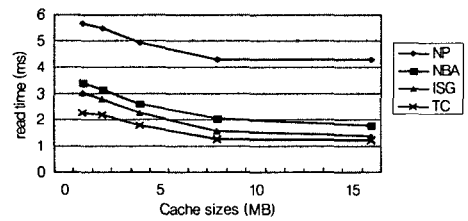
Workload / Characteristics	LIGHT	MEDIUM	HEAVY
Total read size (MB)	29.7	54.7	103.9
Number of total requests	700	1000	1700
Max. request rate (MB/s)	11.2	19.2	32
Avg. read size per request (kB)	48.4	56	62.6
Radom ratio (%)	13.2	23.8	3.76
Number of processes	3	4	5

표 1. 실험에 사용된 작업 부하

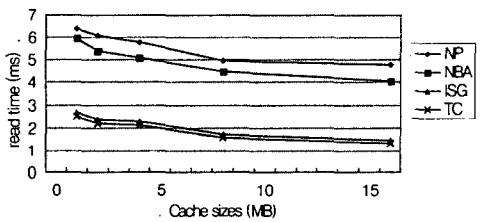
6.3 결과

제안한 선반입 기법의 성능을 비교하기 위해 NP(No Prefetcing), NBA(N-Block Ahead), ISC(Interval-Size-Graph)와 같은 기존의 선반입 기법 또한 적용하여 비교하였다. NP는 선반입을 하지 않는 경우이며, NBA는 요청받은 블록 다음의 N개의 블록을 선반입 하며, ISG는 주어진 패턴을 Graph형태로 유지하여 다음 블록을 예측하는 알고리즘이다.

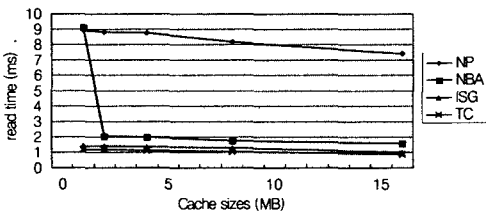
실험은 각 선반입 기법에 대하여, 다양한 작업 부하를 적용하여, 1MB에서 16MB로 버커 캐쉬의 크기를 변경시켜가면서 데이터를 읽는 평균 시간을 측정하였다.



(a) LIGHT



(b) MEDIUM



(C) HEAVY

그림 5. 요청 당 평균 읽기 시간

일반적으로 파일 시스템의 성능측정 방법으로 캐쉬 히트율을 사용하지만, 보다 실제적인 성능의 측정을 위해서는 총 입출력 시간을 측정하였다. 위의 결과에서도 알 수 있듯이, 다양한 작업 부하에서 테이블 비교 선반입 기법이 ISG나 NBA에 비해 우수한 성능을 보여주고 있다. 이것은 ISG에 비해 단순한 알고리즘을 적용하면서도, 다양한 패턴의 접근을 예측할 수 있으며, 현재의 입출력 상황을 고려하여 선반입 여부를 결정하기 때문에, 선반입으로 입출력 부하가 추가되는 것을 방지하여, 선반입으로 전체 파일 시스템의 성능 저하를 막아준다.

7. 결론

병렬 파일시스템은 컴퓨팅 환경에서 비교적 느린 발전을 보이는 디스크의 속도를 보완하기 위해서 병렬성을 파일시스템에 도입한 것이다. 이러한 파일 시스템의 성능을 결정하는 요소로는 캐쉉 기법과 선반입 기법을 들 수 있다.

본 논문에서는 새로운 선반입 기법을 제안한다. 이 선반입 기법은 과학계산용 응용의 접근 패턴에 강하면서 알고리즘이 간단하여 시스템에 부하가 적은 블록 예측 기법을 도입하였으며, 현재의 입출력 상황을 고려하기 위해 가용 입출력 대역폭을 계산하여 선반입 여부와 시기를 결정하여 선반입이 시스템에 부하가 되는 것을 방지하는 선반입 정책을 적용하였다.

시스템에 구현하여 실험한 결과 제안한 테이블 비교 선반입 정책은 시스템에 부하를 주지 않는 간단한 알고리즘으로도 서로 다른 단계의 파일 접근 부하에 대해서 대부분 우수한 성능을 나타내었다. 부하가 과중한 경우도 전체 읽기 시간을 줄일 수 있음을 보여주어 테이블 비교 선반입 정책이 선반입이 시스템의 부하로 작용하는 것을 막아준다는 사실을 알 수 있었다. 이는 선반입 정책이 앞으로 사용될 블록을 예측하는 것뿐만이 아니라 현재의 파일 요청 상황을 파악하여 이를 선반입 정책에 반영할 수 있어야만 모든 파일 요청 부하에 대해서 선반입이 시스템에 유리하게 작용할 수 있음을 보여준다.

6. 참고문헌

[1]. Prasenjit Sarkar and John Hartman, Efficient Cooperative Caching using Hints, Proc. of the USENIX 1996, 2nd Symposium on Systems Design and Implementation, Seattle, Washington, Oct 28-31, 1996.

[2]. J. Griffioen and R. Appleton, Performance Measurements of Automatic Prefetching, Parallel and Distributed Computing Systems, pages 165--170. IEEE, Sept. 1995.

[3]. Toni Cortes, Cooperative Caching and Prefetching in Parallel/Distributed File Systems, PhD thesis, UPC: Universitat Politècnica de Catalunya, Barcelona, Spain, 1997.

[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]. David Kotz and Nils Nieuwejaar, File-Access Characteristics of Parallel Scientific Workloads, *in* IEEE Parallel and Distributed Technology, Spring 1995, pages 51-60. *in* IEEE Parallel and Distributed Technology, Spring 1995, pages 51-60.

[6]. J. Cho, C. Kim, and D. Seo, "A Parallel File System Using Dual Cache Scheme and Prefetching", *The 2000 International Conference on Parallel/Distributed Processing Techniques and Application (PDPTA2000)*, June, 2000