I-node 간의 블록 재배치를 이용한 파일 분할과 재결합 연산의 성능 평가

박현찬, 김영필, 유 혁
고려대학교 컴퓨터학과

e-mail : hcpark@os.korea.ac.kr

Performance evaluation for split and merge operation using block relocation between I-nodes

Hyun-Chan Park, Young-Pil Kim, Chuck Yoo
Dept. of Computer Science and Engineering, Korea University

요 약

파일에 대한 분할과 재결합은 네트워크를 통한 대용량 파일 전송 시에 자주 사용되는 연산이다. 위 연산들은 현재 유저 레벨의 어플리케이션에 의해 제공되고 있어 동일한 데이터를 외부 장치 내에서 복사하는 불필요한 동작을 수행한다. 이러한 단점을 제거하기 위해 커널 레벨의 파일 시스템에 I-node 간의 디스크 블록 재배치를 수행하는 연산을 설계하였다. 그리고 새로운 분할과 재결합 연산을 구현한 파일 시스템 시뮬레이터로 실험을 수행하여 성능을 평가하였다. 결과적으로, 64Mbytes 크기의 파일에 대해 분할 연산은 399 배, 재결합 연산은 682 배의 수행 시간 감소를 보여주었다.

1. 서론

최근 고속 통신의 발달로 인해 인터넷을 통해서 대용량 파일들이 오가는 일이 많아졌다. 그러한 대용량 파일 전송은 보통 전송시의 에러에 의한 재전송을 최소화하기 위해 파일을 분할하여 보내는 경우가 많다.

현재 이러한 분할 기능은 유저 레벨의 어플리케이션들이 담당하고 있다. 이러한 유저 레벨의 분할 기능은 실제 원본 파일의 분할이 아니라 원본 파일의 내용을 여러 파일에 나누어 복사하는 방식으로 구현되고 있다.

이런 구현에서의 주된 단점은 같은 내용을 다시 한 번 디스크에 기록해야 함으로 인해 발생한다. 첫째는 대용량 파일의 복사에 의한 시간 낭비이고, 둘째는 디스크 공간의 불필요한 소모이다. 또 이와는 별도로 어플리케이션 간의 호환성 문제도 발생한다. 분할에 대한 표준이 없이 어플리케이션마다 서로 다른 구현을 하고 있기 때문에 각각의 어플리케이션이 호환 기능을 따로 제공하지 않으면 서로간의 호환성은 없다고 볼 수 있다.

이러한 이유로 인해 커널 레벨인 파일 시스템에서 직접 파일의 분할 기능을 제공한다면 더욱 효율적으로 인터넷 환경에서 대용량 파일 전송을 지원할 수 있다. 이에 따라 파일 시스템 시뮬레이터를 구성하고 그에 분할/재결합 연산을 추가한 후 실험을 통해 그 성능을 측정해보았다.

2. 배경

커다란 용량의 파일을 네트워크를 통해 전송하고자 할 때, 큰 용량 그대로 전송하지 않고 보다 작은 용량의 여러 파일로 분할한 다음 그것을 각각 전송하는 방법이 널리 이용되고 있다. 파일의 분할과 재결합을 수행하는 어플리케이션들을 이용해 먼저 파일을 분할하고 각각의 파일을 네트워크를 통해 전송한 후 받은 측에서 다시 파일들을 결합시켜 본래의 원본 파일을 얻는 방법이다.

이 같은 추가적인 동작이 사용되는 이유는 네트워크를 통한 대용량 파일의 전송 시 발생할 수 있는 에러에 의한 피해를 최소화하고자 함이다. 에러에 의해 전송된 데이터가 손상되거나 변경되어 원본과 같은 데이터를 얻지 못하였을 경우, 사용자는 데이터의 손상된 부분을 판명하기도 힘들고 판명되더라도 그 부분만 다시 요청하여 데이터를 받아 보강하는 동작도 일반적으로 사용되고 있지 않다. 그렇기 때문에 파일 전체를 모두 새로 전송 받아야 하는 경우가 발생한다.

결국 에러는 데이터를 손상시켜 재전송을 요구하고, 이 때 최소한 재전송에 의한 시간 소요를 줄여보고자 분할된 여러 파일들 중 손상을 입은 파일만을 재전송 받도록 하는 것이다. 손상을 입은 파일을 찾아내는 것은 분할 시 원본에 대한 CRC 값을 따로 저장하여 전달한 후 받은 측에서 그 값을 토대로 받은 파일들을 다시 검사하는 방식으로 이루어지고 있다.

또 다른 사용 분야로는 최근 널리 사용되고 있는 P2P 파일 공유 서비스가 있다. P2P 서비스를 위한 어플리케이션들은 큰 용량의 파일을 잘게 분할한 후 각각의 조각 파일을 서로 다른 사용자에게서 받아올 수 있도록 하는 동작을 제공하여 사용자의 네트워크를 최대한으로 사용할 수 있는 가능성을 마련하고 있다. 이런 방식에서도 전송하기 위해 파일을 분할하고 모든 파일의 조각들을 전송 받은 후 다시 재결합하는 연산을 수행하게 된다.

2.1. 유저 레벨의 분할/재결합 방식의 문제점

이러한 현재 상황에서 문제점은 파일의 분할과 재결합을 유저 레벨에서 수행되는 어플리케이션들이 맡고 있는 데서 발생한다. 분할과 재결합을 구현하는 방법을 가장 단순하고 효율적인 방법으로 구상해보면, 분할하고자 하는 파일의 데이터가 실제로 저장되어 있는 저장장치의 블록을 여러 파일에 순서대로 재배치하는 방식을 생각할 수 있다. 재결합하는 방식은 그 반대로 복구하는 방식으로 구현할 수 있을 것이다. 하지만 현재는 유저 레벨에서 커널 내에 있는 I-node 와 디스크 블록에 관한 정보에 접근할 방법이 제공되고 있지 않다.

현재의 어플리케이션들은 위와 같은 커널 서비스의 부재로 인해 다른 방법으로 파일의 분할과 재결합을 구현하고 있는데, 그것은 분할한 파일의 데이터를 순차적으로 읽어 여러 개의 새로운 파일에 그 내용을 복사하는 것이다. 이러한 구현에서는 같은 데이터에 대한 불필요한 디스크 읽기와 쓰기 동작이 소요된다. 같은 내용의 데이터를 단지 전송을 위해 분할하는 것은 파일에 대한 사본이 요구되는 동작이 아니기 때문에 굳이 데이터를 복사할 이유는 없는 것이다. 그리고 동일한 데이터가 중복해서 기록되므로 디스크 공간이 낭비되는 문제도 발생한다.

3. 커널 레벨의 분할/재결합 연산 설계

위와 같이 파일의 분할과 재결합의 유저 레벨에서의 구현과 그에 따른 문제점을 생각해보면, 동일한 데이터에 대한 디스크 복사를 최소화하고자 하는 것이 문제에 대한 근본적인 접근 방법임을 알 수 있다. 이러한 파일의 내용과 표현에 대한 제어는 유저 레벨에서 접근하기에는 한계가 있다. 따라서 커널 레벨에서 이 같은 접근 방법을 구현해보고자 이를 처리하는 연산을 설계해보았다.

커널 레벨에서의 분할/재결합 서비스 지원은 파일 시스템에 그러한 동작을 수행하는 연산들을 추가함으로써 이루어진다. 분할 연산은 분할하고자 하는 I-node 에 대해 해당 I-node 의 데이터를 기록하고 있는 디스크 블록의 리스트를 가져온다. 그리고 분할하고자 하는 개수만큼 새로이 I-node 를 생성하고 앞서의 리스트에 속한 디스크 블록들을 분할하고자 하는 크기만큼 각각의 I-node 에 배치한다. 마지막으로 분할된 본래의 I-node 는 삭제한다. 이는 디스크 블록을 여러 I-node 가 공유하고 있는 상황에서 오는 여러 문제점들을 미연에 방지하기 위함이다.

재결합 연산은 결합하고자 하는 파일들에서 순서대로 디스크 블록 정보를 가져와 새로운 하나의 파일에 모두 배치함으로써 이루어진다. 그리고 분할 연산에서와 같은 이유로 여러 개의 분할되었던 I-node 들은 삭제한다.

위와 같은 두 가지 연산을 파일 시스템에 추가함으로써 하나의 파일을 분할하고 여러 개의 파일을 다시 하나로 결합하기 위해 소요되는 디스크에 대한 I/O 작업을 최소화할 수 있다. 그리고 같은 동작을 위해 호출되어야 하는 시스템 콜의 숫자도 줄어든다. 본래의 구현에서는 읽기와 쓰기 시스템 콜이 파일의 크기에 비례하게 호출되어야 한다. 이에 비해 I/O 작업의 최소화로 얻을 수 있는 작업 시간이 상대적으로 아주 크기 때문에 뚜렷하게 두드러지지는 않지만 이러한 면에서도 새로운 구현이 장점을 가진다는 것을 알 수 있다.

4. 시뮬레이터 설계

이제까지 논의되었던 새로운 분할/재결합 연산의 성능을 평가해보기 위해 EXT2 파일 시스템[1]의 시뮬레이터를 구현하여 본래의 방법과 새로운 방법을 각각 구현하여 성능을 측정해보았다.

4.1. 시뮬레이터 구조

실험을 위해 커널과 유저 영역을 표현하는 시뮬레이터를 구현하였다. 이를 위해 두 개의 프로세스를 사용하여 하나는 파일 시스템을 포함하는 커널로, 다른 하는 사용자의 입력을 받아 커널에 서비스를 요청하는 쉘로 설정하였다. 둘 사이의 통신은 메시지 큐를 이용해 이루어지고 시그널을 사용하여 동기화를 이루었다.

커널 내부에 있는 파일 시스템-이하 SFS(Simple File System)-은 가상으로 만들어진 256 MBytes 의 디스크를 대상으로 EXT2 파일 시스템을 모사하였다. 이 가상 디스크는 하나의 정규 파일로 이루어져있으며, 리눅스의 EXT3 파일 시스템 위에 존재한다. SFS 의 I-node 는 데이터 블록을 표현하기 위해 4 개의 직접 접근 블록 포인터와 1 개의 1 단계 간접 접근 블록 포인터와 1 개의 2 단계 간접 접근 블록 포인터를 갖는다. 각 블록에 대한 주소를 저장하기 위해 4 Bytes 를 사용하고 하나의 데이터 블록은 1024 Bytes 의 정보를 저장할 수 있다. 따라서 하나의 I-node 가 표현할 수 있는 데이터는 총 1024 * (4+256+256*256) Bytes 로 약 64 MBytes 를 조금 넘음을 알 수 있다.

쉘 프로세스는 유저 레벨을 표현하기 위해 작성하

였으며 커널과의 통신에 소요되는 오버헤드를 표현하고자 두 개의 프로세스 간의 통신을 설정하여 시뮬레이트 하였다. 담당하는 일은 사용자의 입력을 받아 커널에 서비스를 요청하는 일이며, 기존 구현에서는 분할/재결합을 수행하는 간단한 기능을 추가로 수행하도록 구현하였다.

4.2. 기존 방법 구현

쉘 프로세스에 읽기, 쓰기, 파일 생성하기의 세 가지 시스템 콜을 이용해 분할/재결합을 수행하는 루틴을 삽입하였다. 최대한 단순한 구조로 작성하여 기본적인 비교를 해보고자 하였다. 분할하고자 하는 파일에서 순차적으로 SFS 의 한 디스크 블록 크기인 1024 Bytes 씩 읽어 생성한 여러 개의 파일에 나누어 쓰는 작업을 수행한다.

4.3. 블록 재배치를 이용한 새로운 방법 구현

커널의 파일 시스템 연산에 분할과 재결합을 담당하는 연산을 추가하여 쉘 프로세스에서 각각 한 번의 시스템 서비스 요청으로 원하는 작업이 수행될 수 있도록 하였다. 내부적인 동작은 I-node 가 가진 디스크 블록 번호를 가져와 다른 I-node 에 재배치하는 것이 핵심이다.

5. 실험 및 결과

위와 같은 설계를 바탕으로 하여 시뮬레이터를 구현하였으며, 이를 이용해 실험을 수행하였다. 그리고 그에 따른 결과를 분석하여 블록 재배치 방법으로 얻을 수 있는 장점을 확인해보고자 하였다.

5.1. 수행 시간 모델

결과를 분석하기에 앞서 각 방법에 따른 결과를 예상해보기 위해 실제 수행하는 연산을 좀 더 세분하여 모델링하면 아래 <수식 1>과 같다.

분할 $T_s(N_s+2N_b)+T_c(N_s)+T_b(2N_b)+T_r(N_b)+T_w(N_b)$

재결합 $T_s(N_s+2N_b)+T_c(1)+T_b(2N_b)+T_r(N_b)+T_w(N_b)$

<본래 방법>

분할 $T_s(1)+T_c(N_s)+T_b(2N_b)$

재결합 $T_s(1)+T_c(1)+T_b(2N_b)$

<블록 재구성을 이용한 방법>

N_s = 분할 갯수
N_b = 블록 갯수
T_s = 시스템 서비스 요청에 의해 소요되는 시간
T_r = 한 블럭에 대한 디스크 읽기 시간
T_w = 한 블럭에 대한 디스크 쓰기 시간
T_c = 파일 생성에 의해 소요되는 시간
T_b = 블록 제어에 의해 소요되는 시간

<수식 1. 각 방법에 따른 수행 시간 모델>

이 모델에 의하면 본래 방법과 블록 재배치를 이용한 방법의 가장 큰 차이는 $2N_b$ 번의 I/O 연산에 소요되는 시간인 T_r 과 T_w 라 할 수 있다. 각 시간은 크게 시스템의 전체적인 속도에 영향을 받는 부분-T_s, T_c, T_b-과, 그리고 I/O 장치에 의해 큰 영향을 받는 부분-T_r, T_w-로 구분할 수 있다. 시스템의 속도와 I/O 장치의 속도는 대체로 무관하고 큰 차이가 있기 때문에 이를 구분하여 분석해야 할 필요가 있다.

블록 재배치를 이용한 새로운 방법의 장점은 I/O 작업을 최소화하여 전체 수행 시간을 줄이는 것이므로 N_b 가 크면 클수록 T_r, T_w 에 의한 영향이 줄어들어 기존의 방법에 비해 보다 짧은 수행시간을 가질 것을 알 수 있다. T_s 또한 기존의 방법에서는 N_b 와 N_s 에 의해 영향을 받지만 새로운 방법에서는 단 한번의 시스템 콜만이 수행되므로 이에 의한 시간 단축 효과도 있다. 그렇지만 이는 일반적으로 T_s 가 T_r, T_w 에 비해 대단히 작다고 여겨지므로 전체 결과에서 주는 영향은 크지 않을 것이라고 볼 수 있다.

5.2. 실험 설계

실험은 각각의 방법에 대해 분할과 재결합 연산을 수행하여 소요시간을 측정하는 것으로 이루어졌다. 각각 1, 2, 4, 8, 16, 32, 64 MB 의 용량을 갖는 파일들을 생성하고 각각을 8 개의 파일로 분할한 후 이를 다시 새로운 하나의 파일로 재결합하는 시간을 측정해보았다. 측정은 사용자의 요청이 시작된 후 모든 동작이 완료하기까지 소요된 CPU 클럭을 측정함으로써 이루어졌다. 그리고 측정한 CPU 클럭을 CPU 의 동작 클럭 속도로 나누어 시간 단위로 변환하였다. 이 같은 측정 간격을 설정한 이유는 실제 사용시에 소요되는 시간을 측정해보고자 함이었으며, 디스크에 대한 I/O 외에 커널에 대한 서비스 요청 시간 또한 측정에 포함하여 그에 따른 영향도 살펴보기 위함이다. 그리고 결과에 대한 분석 또한 실제 사용시에 얼마나 이득을 얻을 수 있는지를 고려하여 이루어졌다.

실험에 의한 모든 결과값은 각각의 경우에 대해 10 번씩 수행한 후 평균값을 취한 것이다. 측정값의 표준 편차는 거의 약 1% 미만이었다.

실험 환경은 Pentium-M 1.4GHz CPU, 256MBytes 메모리, 4200RPM 의 40GBytes 하드 디스크를 가진 노트북 PC 이다. 5400RPM 혹은 7200RPM 을 사용하는 일반 데스크톱 PC 보다 느린 I/O 환경이라 볼 수 있다. 사용한 OS 는 리눅스 커널 버전 2.4.20-8 의 레드햇 9.0 이다.

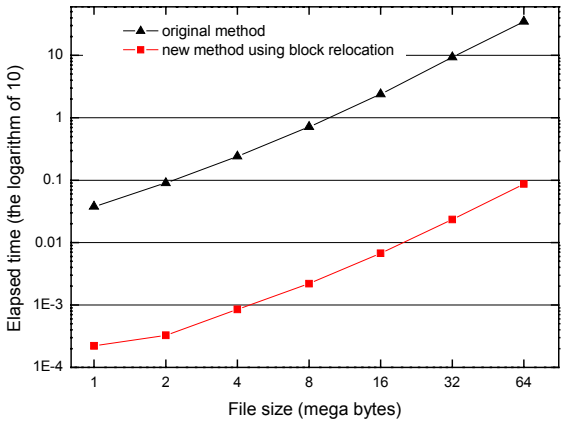
5.3. 결과 및 분석

용량	분할(초)			재결합(초)		
(MB)	본래 방법	블록 재배치	증가율	본래 방법	블록 재배치	증가율
1	0,03764	0,00022	170,51	0,04276	0,00015	281,56
2	0,09006	0,00033	276,64	0,11043	0,00027	412,53
4	0,23975	0,00085	282,11	0,31980	0,00063	507,04
8	0,71682	0,00220	325,81	1,04013	0,00182	570,92
16	2,39521	0,00675	354,81	3,82116	0,00608	628,36
32	9,34733	0,02353	397,25	14,92350	0,02227	669,97
64	34,73050	0,08689	399,69	57,31091	0,08400	682,31

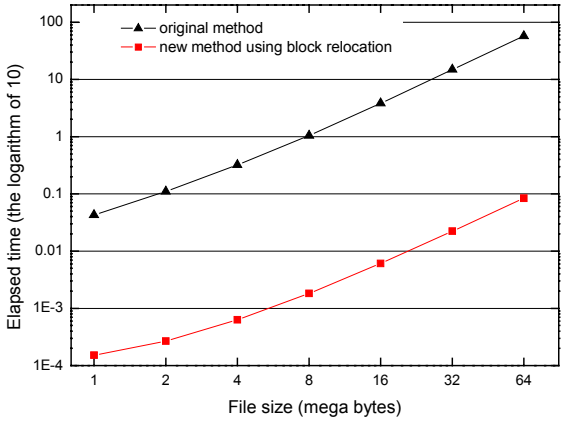
<표 1. 본래 방법과 블록 재배치 방법의 분할, 재결합 소요 시간>

<표 1>은 실험한 결과와 그에 따른 성능 향상의 증

가율을 나타낸 것이다. 절대적인 소요 시간은 실험의 환경에 따라 달라질 수 있다. 따라서 증가율을 살펴보는 것이 보다 정확한 비교를 가능하게 한다고 생각할 수 있다. 그리고 용량에 따른 변화의 추이를 살펴보는 것도 의미를 가진다. 용량이 커짐에 따라 N_b 가 증가하고 전체 시간에서 디스크에 대한 I/O 작업이 차지하는 비중이 커지기 때문이다.



<그림 1. 분할 소요 시간의 로그 단위 그래프>



<그림 2. 재결합 소요 시간의 로그 단위 그래프>

<그림 1>과 <그림 2>는 <표 1>의 결과를 Y 축이 로그 단위인 그래프로 나타낸 것이다. Y 축에서 한 단위 차이는 실제 시간이 10 배 차이가 남을 의미한다. X 축의 값이 지수적으로 증가하고 있으므로 위 그래프의 형태는 <수식 1>의 모든 모델의 수행 시간이 $O(N_b)$ 임을 확인해 주고 있다.

결과는 앞서 예상한 대로 대상 파일의 용량이 증가할수록 보다 높은 증가율을 보이고 있다. 현재 실험 환경은 높은 CPU 클럭에 비해 낮은 I/O 성능을 가진다고 볼 수 있으므로 그에 따른 영향을 생각해본다면 현재 결과는 일반적인 경우보다 높은 T_r, T_w 를 보이고 있다고 추정된다. 그러나 증가율의 수치는 몇 백배나 되므로 그러한 점을 고려하더라도 기존의 방법에 비해 장점이 뚜렷하다.

6. 관련 연구

유닉스의 링크 기능은 파일의 내용을 여러 I-node 가 공유할 수 있도록 해주기 때문에 이러한 동작과 본 논문에서 제시한 방법의 차이점을 명확히 할 필요가 있다.

유닉스는 하드 링크, 심볼릭 링크의 두 가지 파일 연결 기법을 제공한다[2]. 두 기법 모두 하나의 동일한 파일 내용을 디스크 내에서 중복하여 존재하도록 하지 않고 여러 파일로 표현할 수 있다는 점은 본 논문의 기법과 같은 장점을 가진다고 할 수 있다. 하지만 각각의 I-node 가 서로 다른 데이터를 나타내도록 할 수는 없다. 따라서 파일의 분할과 재결합에 이용할 수는 없다. 두 기법은 일반적으로 여러 사용자가 하나의 파일을 공유하기 위해 사용된다.

7. 향후 연구 과제

이 논문은 제안된 새로운 방식의 분할/재결합 연산에 의한 성능을 측정하는 것에 주력하였다. 이 같은 방식이 실제 파일 시스템에 적용되기 위해서는 다른 면에서의 고려도 중요하다. 예를 들면 이와 같은 동작이 파일의 내용을 디스크에 보다 넓게 분포하는 결과를 가져와 성능을 감소시킬 수도 있다. 또 현재는 디스크 블록의 공유에 의한 문제점들을 제거하기 위해 분할과 재결합에 사용된 원본 파일을 삭제하고 있는데, 보다 심도 깊게 이에 대한 고려를 해볼 필요도 있다. 재사용이 필요한 경우 원본을 그대로 유지하는 것이 좋을 수 있기 때문이다.

이러한 고려에 더해 기존의 파일 시스템에 적용하기 위해 각각의 파일 시스템에서 새로운 방법을 구현하는 연구가 필요하다.

8. 결론

파일의 분할과 재결합을 위한 연산을 보다 효율적으로 수행하기 위해 I-node 간의 블록 재배치를 이용한 방법을 제시하고 성능을 평가해보았다. 기존의 방법과 비교해 주된 장점은 유저 레벨에서 분할과 재결합을 수행할 시에 소요되던 디스크의 내용 복사에 의한 I/O 시간이 제거된 점이다. 새로운 분할/재결합 연산을 구현한 시뮬레이터를 이용해 성능을 비교해보았다. 64MB 의 파일을 8 개로 분할하는데 399 배, 다시 하나의 파일로 재결합하는데 682 배의 시간이 감소됨을 확인할 수 있었다.

참고문헌

[1] Dave Poirier, “The Second Extended File System (Internal Layout)”, http://www.freesoftware.fsf.org/ext2-doc/.

[2] Daniel P. Bovet, Marco Cesati, “Understanding the Linux Kernel”, O’REILLY, 2002.

[3] David A. Curry, "Unix System Programming for SVR4", O’REILLY, 2002.