실시간 데이타 저장 시뮬레이션과 성능분석

°김학준, 박기현, 안순신

고려대학교 전자공학과 컴퓨터 네트워크 연구실

Data Placement Technique on Multi-disk & Simulation

Hak Joon Kim, Ki Hyun Park, Sun Shin An

Computer Network Laboratory Dept. of Electronic Engineering

Korea University

요 약

이 논문에서는 먼저 데이타를 디스크에 연속 블럭을 할당해서 저장하는 방식에 관해 간략히 살펴보고 이 방식을 이용해 다수 디스크에 데이타를 저장할 때 제안 된 몇 가지 방식의 특징을 소개, 비교 분석해 보았다. 이를 바탕으로 실제 데이타 저장시 발생할 수 있는 문제들을 고려해서 구체적인 다수 디스크 스케쥴링 방식을 제안하였다. 그리고 이 스케쥴링 방식으로 다수 디스크에 데이타를 저장하는 것을 보여 주기 위해서 가상 디스크를 만들어 실제 데이타가 저장될 때 블럭의 상태를 표시해 주는 시뮬레이션 프로그램을 실행하였다. 마지막으로 이 시뮬레이션 프로그램의 결과를 바탕으로 디스크 스케쥴링에서 고려해야 할 사항들에 대한 내용을 언급하였다

1.서론

갈수록 멀티미디어 데이타처럼 실시간 재생을 필요로 하는 데이타들이 증가됨에 따라 여기에 발맞춰 이와 관련된 여러 분야에서 빠른 데이타 처리를 위한 꾸준한 연구 개발이 진행되고 있다. 하드웨어적으로 마이크로프로세서, 메모리, I/O 칩 등 관련 반도체들의 처리 속도가 빨라지고 주변장치들도 빠른 데이타 처리를 하기 위해 나날이 향상되고 있다. 이런 시점에서 변화하는 하드웨어를 뒷받침 해줄 수 있는 보다 강력한 소프트웨어들이 요구되고 있다. 이렇게 멀티미디어에 관련된 소프트웨어 개발이 더욱 중요시되고 있고 이에 관련된 논문도 많이 나오고 있는 시점에서 이 논문 또한 빠른 데이타 처리, 그 가운데서도 디스크 스케쥴링에 관해 다루고 있다. 이 논문이 다루고 있는 디스크 스케쥴링 방식은 연속 블럭 할당 방식으로 전에도 몇 번 논문으로 소개된 적이 있는데 이것을 다수 디스크에 적용 구체적인 모델을 제시하고 이를 가지고 실제로 시뮬레이션 하는 것이 이 논문의 핵심 내용이라 할 수 있다. 먼저 연속 블럭 할당 방식에 관해 개략적으로 살펴보자.

멀티미디어 데이타를 빠르게 재생하기 위해서는 논리적으로 이웃하는 데이타들을 가능한한 연속된 디스크 블럭에 할당하고 연속된 블럭간의 실린더 간격을 좁혀 디스크의 seektime을 줄여야 한다. 이 원리에 기반을 두고 있는 것이 연속 블럭 할당 방식이다. 이 방식은 사용자로부터 데이타와 데이타의 소비율을 받아서 소비율을 지원할 수 있도록 블럭간의 실린더 간격과 연속된 블럭 수를 결정해서 저장한다. 소비율이 높은 데이타일 경우엔 연속 블럭 내의 디스크 블럭 수가 증가하거나 연속 블럭이 놓인 실린더 사이의 간격이 작아져야 seektime이 줄어들어 소비율을 만족시킬 수 있다.

2.다수 디스크에서 스케쥴링

- T_r: 한 개의 디스크 블럭을 읽는 시간
- $T_{seek\ min}$: 트랙간 이동 거리
- $T_{seek\ max}$: 가장 안쪽과 바깥쪽 실린더간 이동 거리
- T_s: 임의의 블럭 위치로 헤드이동시 필요한 시간
- T_{rotate}: 디스크의 회전 시간
- r_c: 저장시 지정된 데이타의 최대 소비율 (블럭/초)
- r_t: 디스크에서 버퍼로 이동되는 데이타의 재생율(블럭/초)
- S_{max}: 디스크 트랙 수
- T_{total}: seektime을 포함하여 연속 블럭을 읽는데 필요한 시간
- d: 현재 헤드 위치에서 원하는 연속 블럭까지의 트랙 거리
- n: 각 연속 블럭 내의 디스크 블럭 수

같은 용량을 갖는 단일 디스크에 비해 다수 디스크를 사용하면 동일한 시간 내에 더 많은 데이타를 읽을 수 있다. 다수 디스크로 스케쥴링할 때는 모든 디스크 헤더가 동시에 같은 위치를 움직이는 경우와 서로 독립적으로 움직이는 경우로 나뉘어진다. 전자를 동기 디스크라 부르고 후자를 비동기 디스크라 부르기로 하자

2.1.동기 디스크

먼저 디스크 헤더가 동시에 같은 위치를 움직이는 동기된 디스크인 경우를 살펴보자. 이 경우엔 모든 디스크가 동일한 실린더 위에 있으므로 동일한 d 값을 가지나 n 값을 처리하는 방식에 따라 2 가지로 나뉘어진다. 모든 디스크 헤드들이 동일한 실린더에서 같은 n 값 크기의 블럭을 읽는 경우(모든 디스크에서 n 값이 동일)와 동일한 실린더에서 서로 독립적으로 블럭을 읽는 경우(이 경우 디스크마다 제각기 n 값을 가짐)로 각기 다른 재생율을 갖는다. 이를 식으로 계산하면 식 4>식 5>와 같다.

식 4>

$$r_t = \frac{nK}{nT_r + T_{seek\ min} + (d-1)\dfrac{T_{seek\ max} - T_{seek\ min}}{S_{max} - 1}}$$

식 5> $$r_t = \frac{\sum_{i=1}^{k} n_i}{n_{\max} T_r + T_{seek\,min} + (d-1)\frac{T_{seek\,max} - T_{seek\,min}}{S_{max} - 1}} *$$

*: Ji Young Lee. Kihyun Park. Sunshin An. "Realtime Data Structure on Multi-disk". Proceedings of The 23rd KISS Spring Conference. 1996" 참조

식 4. 식 5 모두 단일 디스크로 데이타를 읽는 경우에 비해 식 4의 경우엔 동일 시간당 n배 많은 양의 데이타를 읽을 수 있고 식 5역시 최대 n배 많은 데이타를 읽을 수 있다. 식 4와 식 5의 경우를 비교해 보면 식 5의 경우엔 독립적으로 블럭을 읽어 데이타를 처리할 때 쉬는 디스크들이 발생할 수 있으나 식 4의 경우엔 같은 크기의 연속 블럭을 읽어 쉬는 디스크들이 발생하지 않아 상대적으로 많은 데이타를 읽게 된다. 그러므로 식 4가 식 5에 비해 높은 재생율을 갖는다. 반면에 독립적으로 연속 블럭을 읽는 경우에는 각각의 디스크 헤드들이 처리한 블럭들의 수가 동일할수록 재생율이 같은 크기의 연속 블럭을 읽는 경우의 재생율에 가까워지게 된다.

2.2.비동기 디스크

다음으로 디스크 헤드들이 서로 독립적으로 각각의 실린더 위치에서 움직이는 비동기 디스크의 경우를 보자. 각 디스크 헤드들이 서로 다른 위치에 존재하면서 독립적으로 연속 블럭을 읽어 d 값과 n 값이 디스크마다 별개로 존재한다. 이것의 재생율은 각 디스크들의 재생율 합으로 표시될 수 있고 상대적으로 seektime이 길고 긴 유휴 블럭을 읽는 디스크가 재생율에 많은 영향을 미친다. 이 방식 역시 상대적으로 디스크의 seektime이 길거나 디스크가 긴 유휴 블럭을 읽을 경우엔 다른 디스크들이 쉬는 상황이 발생할 수 있으므로 위의 동기된 디스크들 보다 재생율이 떨어 진다.

재생율 관점에서만 본다면 디스크가 동기되고 같은 크기의 유휴 블럭을 읽는 경우가 가장 효율이 좋으나 파일 크기에 따라선 이 방법을 사용하지 못하는 경우가 발생할 수 있다. 그리고 이 방법은 소프트웨어적으로 모든 디스크에서 같은 실린더 위치에 있는 동일한 크기의 유휴 블럭을 찾기 때문에 디스크의 유휴 블럭이 적어지면 적어질수록 다른 방법보다 연속된 유휴 블럭을 찾는데 많은 시간이 소요된다 또한 d 값이 디스크당 실린더 수에 비해 작을 경우엔 디스크가 거의 다 차 대부분의 실린더에 유휴 블럭이 없고 몇 개의 실린더만 유휴 블럭이 한 두개 있을 경우라도 다른 실린더 상에 유휴 블럭이 남았음에도 유휴 블럭을 찾지 못하는 경우가 발생할 수 있다.

3.알고리즘

3.1 개요

디스크에다 파일을 저장할 때 적은 시간안에 높은 재생율을 얻기 위해서 기본적으로 모든 디스크가 동일한 실린더 상에서 같은 크기의 블럭을 찾아 기록하도록 한다. 그리고 파일이 들어갈 블럭 수가 디스크 수보다 적은 경우엔 한 블럭씩 블럭 수만큼 차례대로 디스크에 저장한다. 즉 파일이 들어갈 블럭 수가 디스크 수보다 클 때 까지는 디스크 다다 동일한 크기의 블럭을 저장한 후 저장한 블럭 크기만큼 파일 크기에서 빼나가다 남은 파일이 들어갈 블럭 수가 디스크 수보다 작아지면 디스크들에 한 블럭씩 저장한다. 이 방법은 동기 디스크에서 우선 모든 디스크가 현재 위치에다 동일한 d 값을 더한 실린더 위에서 같은 n 값의 연속 유휴 블럭을 찾아 처리하도록 하고 처리할 블럭 수가 디스크 보다 작을 경우엔 n 값을 디스크에 따라 1과 0으로 달리해 주는 것으로 2.1 에서 기술한 모든 디스크가 동일한 n 값을 갖는 경우와 독립적으로 n 값을 갖는 경우를 혼합해 사용하는 방법이다.

3.2.알고리즘의 수식 표현

M을 파일이 들어갈 블럭 수라 놓고 K를 디스크 수라 놓으면 디스크에 저장된 블럭의 총 합이 $[M/K]\times K$이 될 때까지는 각 디스크에 동일한 크기의 블럭을 저장하고 파일에 아직 저장되지 못한 나머지 블럭 들이 $(M-[M/K]\times K)$개 남았을 때는 $(M-[M/K]\times K)$개의 디스크에다 1 블럭씩을 저장한다.

즉 블럭의 총 합이 $[M/K]\times K$가 될 때까지는 각 디스크마다 동일한 수의 블럭이 저장되므로 $[M/K]\times K$를 $\sum_{i=1}^{I-1} X_i \times K$(여기서 X_i는 각 단계마다 한 개의 디스크에 저장되는 블럭 수다)라 놓을 수 있고 다음과 같은 식이 성립한다.

$$M/K-1 < \sum_{i=1}^{I-1} X_i \le M/K$$

I-1 번째까지 동일한 수의 블럭을 매 단계 마다 X_i개씩 각 디스크에 저장하고 I번째에 남은 블럭이 발생 했을 경우에는 한 블럭씩$(M-[M/K]\times K)$개의 디스크에 저장한다. 그런데 I번째에선 모든 디스크에다 한 블럭씩 저장할 수 없으므로 스케쥴링을 해 돌아가면서 순서대로 저장해야 하는데 마지막으로 저장된 디스크의 다음 디스크를 기억했다가 다음 번 파일을 저장할 때 기억했던 디스크부터 순서대로 디스크에 저장하는 방법과 랜덤하게 K개 디스크 중 1개를 선택해서 이 디스크부터 순서대로 저장하는 방법이 있다.

첫 번째 방법은 확실히 남은 블럭이 발생 했을 때 파일들을 균등하게 디스크에 저장할 수 있어 디스크 효율을 높일 수 있으나 항상 다음 번 디스크를 기억하고 있어야 하므로 번거로울 수 있다. 두 번째 방법은 첫 번째 방법에 비해 균등하게 디스크에다 저장할 순 없지만 간단히 구현할 수 있다.

위에서 파일을 디스크에 저장할 때는 I-1 번째까지 각 단계마다 식 4의 재생율 r_{ti}로 저장되나 남는 블럭이 발생하면 재생율은 식 5에서 n_{max}이 1로 $\sum_{i=1}^{k} n_i$이 $(M-[M/K]\times K)$로 바뀌진 식 6처럼 된다.

식 6> $$r_t = \frac{M-[M/K]\times K}{Tr + T_{seek\,min} + (d-1)\frac{T_{seek\,max} - T_{seek\,min}}{S_{min} - 1}}$$

그러므로 파일을 저장할 때 I-1 번째까지는 식 4에 의해 d와 n 값이 정해지나 마지막에 남는 블럭이 발생할 경우엔 I번째에서 식 6에 의해 d와 n 값이 정해지게 된다.

4.시뮬레이션

위에서 제안된 알고리즘으로 가상으로 다수 디스크를 만들어 시뮬레이션을 해보았다. 디스크 수, 실린더 수, 실린더당 트랙 수, 트랙당 섹터 수, 섹터당 바이트 수, 블럭당 바이트 수를 입력 값으로 주면 이 파라미터에 해당하는 가상 디스크를 텍스트 파일로 만들어 준다. 가상 디스크는 실제 데이터가 저장되 있는 것이 아니고 4바이트가 실제 디스크의 1 블럭에 해당되 실제 디스크의 해당하는 블럭에 데이터가 저장되어 있는 지의 여부를 아스키 코드 'F'(Free), 'U'(Used)로 표시해 준다. 이 가상 디스크에다 위의 알고리즘로 가상으로 데이타를 저장해 보았다.

4.1.구조

프로그램은 크게 hard_init( ), write_cblk( ), find_cblk_cyl( ), find_cyls( ), write_file_dsk( )로 나뉘어 진다. hard_init 함수는 실제 디스크의 디스크 수, 실린더 수, 실린더 당 트랙 수, 트랙당 섹터 수, 섹터 당 바이트 수, 블럭 당 바이트 수에 따라 가상 디스크를 만들고 모든 블럭을 'F'로 초기화 한다.

write_cblk 함수: 가상 하드에 접근하는 함수로 사용된 연속 럭을 'U'로 표시해 준다.

find_cblk_cyl 함수: find_cyls 함수에 의해 불려지고 디스크의 실린더에 접근하는 함수로 실린더 위치와 연속 블럭 수를 인자로 받아 여기에 해당하는 블럭 위치를 찾아 리턴해 준다.

find_cyls 함수: wrtie_file 함수에 의해 불려지고 d 값과 n 값을 인자로 받는다. fsize 가 디스크 수보다 클 경우엔 fsize 를 디스크 수로 나눠 몫에 해당하는 연속 블럭을 find_cblk_cyls 함수를 통해 찾아 본다. 이때 find_cyls 함수는 fsize 가 디스크 수보다 작을 경우에는 식 5 에 의해 n 값을 계산하고 1 블럭씩을 find_cblk_cyls 함수를 통해 찾아 본다. 블럭을 찾았을 경우에는 블럭 위치를 write_file 함수에게 리턴한다.

wirte_file_dsk 함수: 파일이 디스크에 저장 됨을 가상 디스크에 표시해주는 함수로 사용자가 부르게 된다. 파일 크기를 인자로 받으며 find_cyls 함수를 불러 현재의 실린더에서 d 만큼 더한 위치와 뺀 위치에서 n 이상인 유휴 블럭을 리턴받아 디스크에 기록하고 fsize 에서 기록한 만큼 감해준다. 파일을 다 저장할 때까지 이 작업을 반복한다. 아래는 이 함수의 알고리즘을 보여주고 있다.

```
write_file_dsk( int fsize )
{
  d 값과 n 값을 결정한다;
  while( fsize > 0 ){

      find_cyls( ); /* 각 디스크의 동일 위치 실린더에서 연속된 유휴 블럭을 찾는다 */
      if ( find_cyls( )가 디스크들의 실린더에서 연속된 유휴 블럭을 찾았을 경우){
          write_cblk();     /* 가상 디스크에 연속 유휴 블럭들을 'U'로 표시해 준다 */
          fsize 에서 기록한 블럭 크기만큼 감해준다;
      else {/* 연속 유휴 블럭을 찾지 못했을 경우엔 d 값을 1 줄이고 n 값을 결정해 준다 */
          if (--d==-1) exit(0); if ( (n=det_n(d) < 1 ) n=1;
          /* det_n(d): 식 4 에 의해 d 값에 따른 n 값을 결정해 준다 */
      }
  }
}
```

write_file 함수에서 처음에 d 값과 n 값을 결정할 때 d 값은 현 위치에서 떨어진 실린더 거리이므로 디스크당 실린더 수보다 클 수 없고 n 값은 연속된 블럭 수이므로 트랙당 블럭 수보다 클 수 없다. 그러므로 먼저 d 값을 디스크당 실린더 수로 놓고 식 4 에 의해 n 값을 구하든지 먼저 n 값을 트랙 당 블럭 수로 잡고 d 값을 구한다. 그런데 식 4 가 파일의 재생율을 만족시키지 못하는 경우가 있다. 먼저 d 와 n 값의 관계는 재생율 r_t 를 식 4 나 5 에 대입하면 양의 기울기를 갖고 y 절편이 0 보다 작은 직선이 된다.

이 때 x 절편이 트랙당 블럭 수를 초과할 경우는 d 값을 0 으로 줄여도 재생율을 만족시키는 연속 블럭이 없으므로 디스크의 하드웨어적 특성인 디스크 헤드의 트랙간 이동 시간 $T_{seek\ min}$ 를 줄여 x 절편을 트랙당 블럭 수 안으로 들어오도록 해줘야 이런 재생율을 갖는 파일을 저장할 수 있다.

d 값은 프로그램이 exit(0)될 때 디스크 상에 유휴 블럭으로 남는 블럭 양의 많고 적음에 많은 영향을 주는데 d 값과 유휴 블럭으로 남는 블럭 양의 관계가 뒤에 시뮬레이션 결과로 나와 있다. write_file 은 d 값이 0 이 아닌 -1 이 되었을 때 exit(0)하게 되어 있는데 이는 d 값을 줄여가면서 유휴 블럭을 찾다가 찾지 못해 현 실린더 위치까지 오게 되면 현 실린더에서 찾기 위해서다.

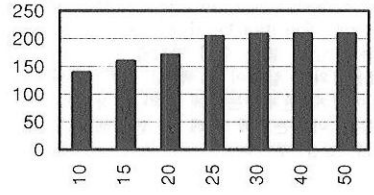
5.성능분석

성능 분석을 위해 디스크 수 4 개, 디스크 당 실린더 수 50 개, 실린더 당 트랙 수 32 개, 트랙 당 섹터 수 64 개, 세터 당 바이트 512Byte, 블럭 당 바이트 8192Byte, Tseekmin 10^{-4} sec, T_r $2*10^{-4}$ sec, $\frac{T_{seek\ max} - T_{seek\ min}}{S_{max} - 1}$ $2.5*10^{-4}$ 인 209M 가상 디스크를 사용하였다. 이 가상 디스크에다 초당 1000 블럭의 최대 소비율을 지닌 크기가 1k-10M byte 되는 파일들을 랜덤하게 디스크가 다 찰 때까지 저장 하였다.

5.1.적정 d 값 시뮬레이션

d 값을 변화 시키면서 가상으로 데이타를 저장시킬 때 디스크 용량의 몇 퍼센트까지 채울 수 있는지를 d 값을 10, 20, 30, 40, 50 으로 증가시키면서 디스크 용량의 변화를 보았다

x축:d값 y축:저장량(Mbyte)



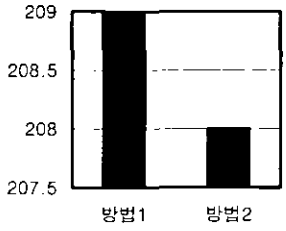
d 값이 디스크 당 실린더 수에 가까워 질수록 디스크 용량의 대부분이 채워짐을 알 수 있었다.d 값이 20 에서 25 사이에서 디스크 용량이 갑자기 증가했고 40 과 50 에서는 전 디스크 용량이 채워짐을 보이고 있다. 이 외의 시뮬레이션에서도 일정 d 값 변화에 갑작스럽게 용량이 증가했음을 관찰할 수 있었다. 파일을 저장할 때 d 값을 트랙 당 실린더 수에 가깝게 놓는 것이 디스크 전체를 보았을 때 유리하다.

5.2.방향 스케쥴링 시뮬레이션

데이타를 저장할 때 현 헤드 위치에서 d 값만큼 더하고 뺀 위치다 저장해야 하는데 두 곳 모두 저장할 연속 블럭들이 존재할 경우에는 어느 쪽에다 저장할 지를 지정해주어야 한다. 이것을 방향 스케쥴링이라 부르고 몇가지 방식이 있을 수 있다. 첫째로 방향을 무조건 안쪽 또는 바깥쪽 중에 한 방향으로 결정해 주는 경우가 있고 둘째로 랜덤하게 그때 그때마다 결정해 주는 방식이 있다 각 방식에 따라 저장된 디스크 용량의 변화를 보았다.

아래 그림에서 볼 수 있듯이 첫번째 방법은 디스크를 다 채웠으나 두번째 방법은 다소 빈 블럭을 남겼다. 이것은 두번째 방법이 한 쪽 방향에다 우선 순위를 주므로 디스크의 안쪽 또는 바깥쪽에 몰릴 경우 벗어나지 못하고 끝내기 때문이

다.하지만 두번째 방법 또한 대부분의 디스크를 채웠음을 보여준다.



y 축:저장량(Mbyte)

6.결론

연속 블럭 할당 방식은 저장된 데이타의 위치를 검색하는데 관점을 둔 디스크 스케쥴링 과는 달리 적정 d 값과 n 값에 해당하는 위치에 데이타를 저장하는데 관점을 두고 있다. 본 논문에서도 가상으로 데이타를 저장하는 알고리즘과 시뮬레이션에 촛점을 맞췄다. 알고리즘은 동기 디스크에서 2 가지 방식을 상황에 따라 사용했는데 여기다 비동기 디스크의 경우도 첨가해서 사용한다면 좀 더 나은 효율을 갖게 될 것이다. 스케쥴링시 그다지 빠른 속도를 내지 못했는데 이점도 보완해야 할 부분 중 한가지다. 4 장에서 n 값과 d 값의 관계를 직선으로 표시할 때 이 직선이 x 축으로 트랙 당 블럭 수, y 축으로 디스크 당 실린더 수로 이루어진 사각형 안에 어떤 식으로 위치하느냐에 따라 데이타 저장 속도라든지 디스크에서 사용되지 못하고 남는 블럭 수에 영향을 미침을 볼 수 있었다. 직선의 기울기와 y 절편이 트랙 당 블럭 수와 디스크 당 실린더 수와 어떻게 관계하는지에 따라 저장 속도나 저장 능력에 어떤 영향을 미치는지에 대해 더 연구해 보면 흥미로운 결과를 얻을 수 있을 것이다.

참고 문헌

[1] P. Venkat Rangan, Harrick M. Vin, "Efficient Storage Techniques for Digital Continuous Multimedia", IEEE Trans on Knowledge and Data Engineering. Vol. 5, No. 4. Aug. 1993. pp1041-4347
[2] D. James Gemmel, Jiawei Han" Delay-Sensitive Multimedia on Disks", IEEE multimedia. 1994. pp.56-67
[3] Ralf Steinmetz, "Multimedia file systems survey: approaches for continuous media disk scheduling", Computer communications Vol. 18. No. 3. Mar. 1995. pp. 133-144
[4] David P. Anderson, Yoshitomo Dwawa, Ramesh Govindan, "A File System Continuous Media", ACM Transactions on Computer Systems. Vol. 10. No. 4. Nov. 1992. pp. 311-337
[5] Maesil Seo, Cheeha Kim, "Allocation Strategies of Multimedia Data on Disk Arrays", Computer communications. 1995. Vol. 18. No. 3. pp. 185-191
[6] Yeonjung Kim, Myungsook Jin, Kihyun Park, Sunshin An, "A Placement Algorithm for retrieving Realtime Multimedia Data", Proceedings of The 22nd KISS Fall Conference. 1995
[7] Kihyun Park, ByungKwen Song, Sunshin An, "Realtime Multimedia Placement Algorithm and its File System", IASTED'96 on Networks. 1996
[8] Ji Young Lee, Kihyun Park, Sunshin An, "Realtime Data Structure on Multi-disk", Proceedings of The 23rd KISS Spring Conference. 1996"