NAND 플래시 메모리 파일 시스템의 빠른 연산을 위한 설계 및 성능 평가

진종원°, 이태훈, 정기동
부산대학교 컴퓨터공학과
e-mail:{waller°, withsoul}@melon.cs.pusan.ac.kr, kdchung@pusan.ac.kr

Design and Evaluation of Fast Operation Method for NAND Flash Memory File System

Jong-Won Jin°, Tae-Hoon Lee, Ki-Dong Chung
Dept. of Computer Engineering, Pusan National University

1. 서론

플래시 메모리는 비휘발성, 저 전력, 빠른 입출력, 충격에 강함 등과 같은 많은 장점을 가지고 있으며 모바일 기기에서의 저장 매체로 사용이 증가 되고 있다. 또한 하드디스크가 데이터의 위치에 따른 헤드의 탐색 거리에 따라 성능에 큰 영향을 주지만 플래시 메모리의 경우 위치에 상관없는 빠른 입출력 속도를 가지고 있다. 하지만 제자리 덮어쓰기가 불가능하고 지움 연산의 단위가 크다는 제약 및 블록의 지움 횟수 제한이 있다[1]. 이러한 제약을 극복하기 위해 YAFFS[2]와 같은 로그 구조 기반의 플래시 파일 시스템들이 개발 되었다.

그러나 YAFFS는 로그 구조를 기반으로 하였기 때문에 공간 할당을 위해서는 순차적인 방법으로 블록 상태를 검색하여 여유 블록을 할당한다. 순차적인 블록 상태를 검색하여 할당하는 방법은 플래시 메모리 사용량이 높아 여유 블록이 연속적이지 않을 경우 여유 블록 할당에 많은 시간이 소모 된다. 또한 블록 지움 횟수에 제한이 있어 블록 할당을 수행할 때에 이를 고려하여야 한다. 마찬가지로 블록 지움 연산 수행 시에도 지움 대상 블록을 순차적으로 검사하는 작업으로 인해 많은 시간을 소모하게 된다. 그리고 블록 지움 연산 수행을 파일객체 쓰기/갱신 작업 시에 페이지 단위로 지움 대상 블록을 검사함으로 인해 검사 시간에 많은 비용을 소모하게 되고, 플래시 메모리의 사용공간을 고려하지 않고 수행함으로 인해 시스템에 블록 지움 연산 수행을 위한 성능 저하가 있다.

이러한 문제점을 해결하기 위해 본 논문에서는 로그 구조 기반의 NAND 플래시 메모리 파일시스템의 빠른 연산을 위한 관리 기법을 제안하고자 한다.

2. 본론

2.1. YAFFS

NAND 플래시 메모리를 위한 대표적인 파일 시스템으로는 YAFFS(Yet Another Flash Filing System)가 있다. YAFFS는 로그 구조를 기반으로 하여 플래시 메모리에 대한 갱신 연산을 추가 연산으로 변형하여 처리하는 외부 갱신 기법을 사용한다. 이를 통해 플래시 메모리의 제자리 덮어쓰기가 되지 않는 문제점을 해결하였다[3].

그림 1은 YAFFS가 사용하는 자료 구조를 보여주고 있다. RAM상의 allocationBlock과 allocation페이지는 현재 사용 중인 블록과 블록 내의 페이지 번호를 나타낸다. Block_Info 구조체는 마운팅 과정에서 플래시 메모리를 읽어 RAM상에 구축된다. Block_Info 구조체는 각 블록의 유효한 페이지 수, 유효하지 않은 페이지 수, 상태 정보 등을 나타낸다. 이러한 구조로 인해 플래시 메모리의 사용량이 높아 여유 블록의 적고 플래시 메모리 상에 많이 흩어져 있는 경우, 공간 할당 시 여유 블록 검색에 많은 시간이 소요된다. 또한 지움 횟수에 대한 고려 없이 순차적인 할당을 수행함으로 인해 플래시 메모리의 균등한 사용을 하기에 힘든 구조를 가지고 있다. 그리고 블록 지움 연산 수행 시 지움 대상블록을 순차적으로 검사하는 작업을 수행하는 것으로 인해 많은 시간을 소모하게 된다. 또한 YAFFS 파일시스템은 블록 지움 연산 수행을 파일객체 쓰기/갱신 작업 시에 페이지단위로 지움 대상 블록을 검사함으로 인해 검사 시간에 많은 비용을 소모하게 된다.

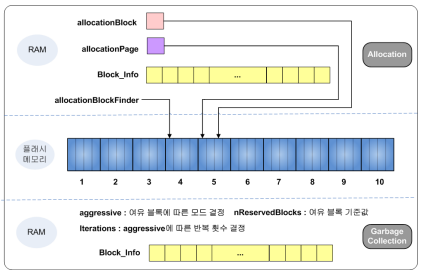

그림 1 YAFFS 자료구조

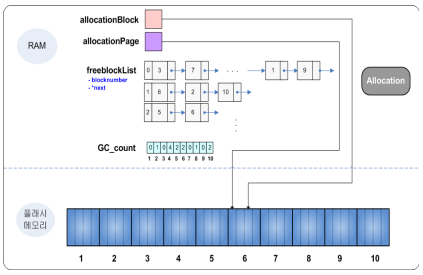

그림 2 균등 사용 및 빠른 연산을 위한 구조

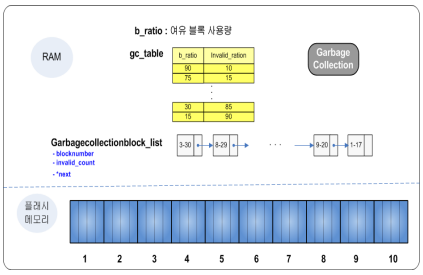

그림 3 블록 지움 연산 향상을 위한 구조

2.2 제안하는 관리 기법

2.2.1. 균등 사용을 고려한 빠른 할당 기법

YAFFS 파일시스템의 경우 여유블록 할당을 위해서 각 블록의 상태를 순차적으로 Block_Info라는 구조체에서 읽어 온다. 이렇게 순차적인 구조는 블록의 여유 공간이 부족하여 여유블록 간의 거리가 멀어지게 되면 여유블록을 찾기 위해 불필요한 자원을 사용하게 된다. 이러한 순차적인 검색의 문제를 해결하기 위해 여유 블록 정보를 연결 리스트를 사용하여 유지한다.

그림 2은 공간 할당을 위한 여유 블록 연결 리스트를 보여주고 있다. 연결 리스트의 각 노드는 다음 여유 블록의 위치와 다음 노드를 가리키는 포인터로 구성된다. 그리고 여유 블록 연결 리스트는 블록 지움 연산을 수행한 후에 블록 지움 연산 수행 횟수를 기록하여 블록 지움 연산 수행 횟수별 연결 리스트를 유지한다. 이 정보들을 이용하여 블록 할당 요청의 경우 블록 지움 연산이 작은 연결 리스트의 여유 블록을 먼저 할당하게 함으로서 균등한 블록 사용을 보장해 줄 수 있게 된다.

위와 같은 방법으로 제안한 정보를 유지할 경우 연결 리스트를 위한 추가적인 메모리 사용이 필요하게 되지만 플래시 메모리 용량이 커지고 사용량이 많이 질수록 선형적인 구조에 비해 연결 리스트로의 정보 유지가 효율적게 된다. 또한 연결 리스트를

관리하는데 있어 블록 지움 횟수별로 따로 리스트를 유지하는 것으로 우선순위별 리스트 구조를 구성함으로서 삽입/삭제 시에 효율적인 관리를 제공 할 수 있다.

2.2.2. 플래시 메모리 블록 지움 연산의 개선

YAFFS 파일시스템의 경우 여유 블록 확보 작업을 수행 하는 경우는 파일의 정보를 생성/갱신 할 경우에 페이지 단위로 지움 대상 블록을 검사하게 된다. 이때 여유 블록이 기존 확보되어야 할 최소의 값인 nReservedBlocks값 보다 작아 충분한 공간이 없을 시에는 전체 블록을 검사하여 가장 dirtiest한 블록을 찾아서 블록 지움 연산을 수행하게 되고, 충분한 공간이 있을 때에는 특정 iteration값을 통해서 매 페이지 쓰기 연산을 수행 할 때마다 iteration으로 지정된 만큼의 블록을 검사하여 지움 대상 블록을 검사한다. 이러한 방법으로의 지움 대상 블록 검사는 불필요한 오버헤드를 생기게 하고 또한 충분한 공간이 없는 상황에서의 지움 대상 블록의 검사는 전체 블록검사라는 큰 부하가 시스템에 걸리게 된다. 이러한 성능 저하를 가져오게 되는 문제를 해결하기 위하여 본 논문은 블록 지움 연산의 대상이 되는 블록의 무효화 된 페이지의 수를 유지하면서 무효화 페이지가 많은 블록을 우선 배치하여 연결 리스트 구조로 관리한다. 그림 3은 블록 지움 연산을 효율적으로 수행하기 위한 구조를 지움 대상 블록 연결 리스트의 구조를 보여 주고 있다. 연결 리스트의 각 노드는 다음 지움 대상 블록의 위치, 다음 노드를 가리키는 포인터와 블록에서의 불필요한 페이지 정보의 수로 구성된다. 또한 지움 연산을 수행하는데 있어 언제 수행하느냐가 중요한 문제가 되는데 기존의 YAFFS 파일시스템은 파일객체 쓰기/갱신 작업 시에 페이지단위로 지움 대상 블록을 검사함으로 인해 검사 시간에 많은 비용을 소모하게 된다. 그러나 제안한 방법으로의 관리를 통해 불필요한 페이지가 많은 블록을 먼저 제거 할 수 있게 함으로서 불필요한 검색 시간을 제거 하였다. 이렇게 구조를 변경 하게 되었을 때 기존의 YAFFS 파일시스템과 달리 검사를 수행하는 것이 아니라 여유 블록 확보가 필요한 시점에 블록 지움 연산을 통해 확보가 가능하다. 본 논문에서는 이 시점을 새로운 블록을 할당 받을 때에 플래시 메모리의 사용량을 보고 사용량별 무효화 회수의 제한값을 가지는 테이블을 유지하여 플래시 메모리의 사용량이 작을 때에는 해당 블록이 무효화된 페이지가 많은 블록에 대해서만 지움 연산을 수행하고 플래시 메모리 사용량이 많아지면 무효화에 대한 제한 수치값을 감소하게 함으로서 여유 블록 확보를 더 많이 할 수 있도록 하였다. 앞서 설명한 플래시 메모리 사용량별 무효화된 페이지가 많은 블록을 선택하는 방법에 대한 값은 임의로 지정하였다.

3. 결론

제안하는 기법들을 실제 실험을 통해 성능을 비교하고 분석하였다. 실험을 위해 휴인스에서 개발된 PXA255-III 임베디드 보드를 사용하였다. 삼성에서 개발된 64MB NAND 플래시 메모리를 사용하였으며 YAFFS에 제안한 기법들을 구현하여 실험하였다. 블록 할당 성능은 플래시 메모리에서 여유 블록이 흩어져 있는 상태에 영향을 받으므로 여유 블록의 분포를 조정하여 실험을 수행하였다. 그리고 블록 지움 연산의 성능을 확인은 블록의 지움 연산 함수를 호출했을 때 수행 시간, 제안한 기법과 기존의 기법의 호출 빈도수, 플래시 메모리의 사용량의 증가에 따른 블록 지움 연산 수행 시간을 비교 하였다.

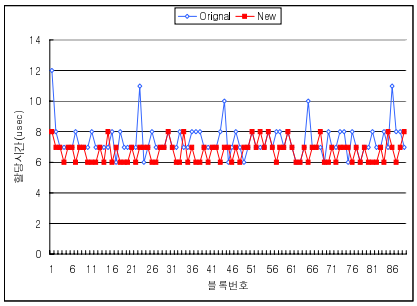

그림 4 할당 시간 비교

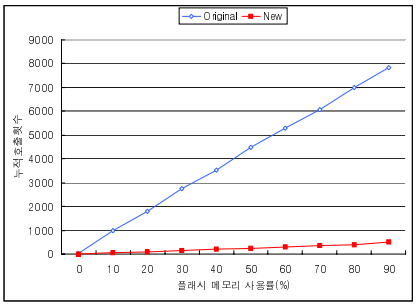

그림 5 지움 연산 누적호출횟수 비교

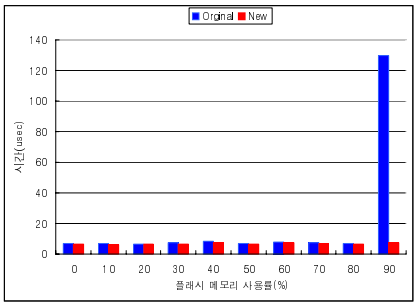

그림 6 사용율별 지움 연산 수행 시간 비교

그림 4는 플래시 메모리에 여유 블록을 일정 블록 단위로 설정을 하여 블록 할당을 위한 성능을 측정하였다. 본 논문에서는 20개 블록 단위로 여유블록을 설정하였다. 보는 것과 같이 기존의 YAFFS의 경우 여유 블록이 연속적이지 않은 경우에 할당시간이 증가하는 것을 확인 할 수 있다. 이러한 설정으로 기존의 방법에 비해 평균 10%의 성능 향상을 보였다. 플래시 메모리의 용량이 커지고 사용량이 많아질수록 여유블록 할당을 위한 블록간 거리에 대한 검색 오버헤드가 커지게 되므로 더 많은 시간을 소요하게 된다. 그러므로 제안한 기법은 NAND 플래시 메모리의 크기에 비례하여 성능향상을 기대 할 수 있다.

그림 5는 사용량에 따른 지움 대상 블록을 호출하는 누적횟수를 나타낸 것이다. 기존의 방법은 지움 연산을 수행하기 위해서 지움 연산의 대상 블록이 되는 블록을 호출하여 수행함으로 인해 지움 대상 블록을 검사하게 된다. 이렇게 지움 대상 블록을 검사하는 작업은 실제 지움 연산을 수행하는 횟수에 비해 빈번하게 이루어진다. 기존의 방법에서는 지움 대상 블록에 대한 정보를 Block_info 정보에서 각각의 블록을 읽어 확인해야 하는 구조로 되어 있어 지움 대상 블록을 확인하기 위한 호출이 빈번하게 이루어진다. 그러나 제안한 방법의 경우 Block_info에서 정보를 읽어 오는 것이 아니라 무효한 페이지 수에 대한 정보를 통한 연결 리스트로 정보를 유지하고 있으므로 블록 할당을 하면서 블록 지움 대상 블록에 대한 확인을 하므로 블록 호출 횟수가 기존의 YAFFS에 비해 약 6%에 불과 하였다.

그림 6은 플래시 메모리의 사용률별 블록 지움 대상 블록을 선택하는 시간을 나타내고 있는데 기존의 YAFFS의 경우 사용률이 높아져 충분한 공간이 없게 되면 전체공간을 검사하는 상태로 되어 지움 대상 블록을 선택하는데 전체공간을 검사하는 시간이 걸리게 된다. 그러나 제안한 기법은 플래시 메모리 사용량이 증가하는 것과 관계없이 일정한 시간이 걸리는 것을 볼 수 있다. 그림 6의 플래시 메모리 사용률별 시간 비교는 지움 대상 블록을 선택하는 한번의 시간을 나타내고 있는 것으로 실제 시스템에서는 그림 5에서 보는 것과 같이 지움 연산을 위한 호출 횟수가 기존 시스템이 많기 때문에 시스템 부하는 기존 시스템이 그만큼 커지게 된다.

본 논문은 로그 구조 기반의 NAND 플래시 파일시스템을 위한 효율적인 관리 기법들을 제안하였다. 빠른 할당을 위해 연결 리스트 구조를 이용하였고 이에 플래시 메모리의 블록 지움 횟수를 포함시켜 균등화된 사용이 가능하게 하였다. 또한 플래시 메모리 블록 지움 연산에 대한 연결 리스트를 유지하고 블록 지움 연산이 필요한 블록을 검사하는 과정의 오버헤드를 제거하여 플래시 메모리의 사용량에 관계없이 빠른 여유 블록 확보를 할 수 있게 하였다.

참고문헌

[1] 백승재, 최종무, "플래시 메모리 파일 시스템을 위한 순수도 기반 페이지 할당 기법에 대한 연구", 정보처리학회논문지 A 제13-A권 제5호, 2006.10
[2] Yaffs Spec, http://www.aleph1.co.uk/taxonomy/term/31
[3] 박송화, 이태훈, 정기동, "임베디드 기기를 위한 NAND 플래시 파일 시스템의 설계", 한국컴퓨터종합학술대회, 논문집(A) 제 33권 1호, pp.151-153, 2006.