NAND 플래시 파일 시스템을 위한 적응적인 가비지 컬렉션 기법

이상기*, 이태훈*, 정기동*

*부산대학교 컴퓨터공학과

e-mail:{nick, withsoul}@melon.cs.pusan.ac.kr,
kdchung@pusan.ac.kr

Adaptable Garbage Collection Techinque For NAND Flash File System

Sang-Gi Lee*, Tae-Hun Lee*, Ki-Dong Jung*

*Dept of Computer Science, Pusan National University

요 약

플래시 메모리는 강한 내구성과 소형화, 대용량화라는 특성 때문에 임베디드 시스템 및 관련 기기에서 널리 사용되고 있다. 플래시 메모리는 지움 횟수 제한이 있고 제자리 업데이트 시 지움 연산이 선행되어야 한다. 또한, 쓰기 단위에 비해 지움 단위가 커서 연산시간이 많이 걸리며, 지움 대상 블록이 많은 경우에는 쓰기 연산 지연의 원인이 된다. 본 논문에서는 이러한 쓰기 연산 지연을 예방하기 위하여 쓰기 연산량에 따라 지움 연산을 제어하는 효율적인 가비지 컬렉션 기법을 제안하고, YAFFS에 구현하여 성능 평가를 하였다.

1. 서 론

임베디드 시스템(Embedded System)은 항공, 가전기기, 자동차, 휴대폰, 공장자동화 등 다양한 분야의 산업에서 사용되고 있다. 임베디드 시스템의 특성상 충격에 약하고 크기가 큰 하드 디스크는 적절하지 않다. 이와 같은 이유로 최근 임베디드 시스템에서는 크기가 작고 충격에 강한 플래시 메모리를 주로 사용한다. 플래시 메모리는 전기적으로 지우고 쓰기가 가능하며 랜덤(Random) 접근이 가능하다. 플래시 메모리는 NOR와 NAND형 두 가지가 있으며, NAND 플래시 메모리는 NOR에 비해 읽기 속도가 빠르며 직접도가 높아 대용량인 임베디드 시스템에서 주로 사용된다[2, 3].

<표 1> NAND 플래시 메모리의 특성(SAMSUNG K9F5608X0D)

특 징	값
Random Access	15 ㎲
Serial Page Access	50 ns
Program time	200 ㎲
Block Erase Time	2 ms
Page Program	(512 + 16)Byte
Block Erase	(16K + 512)Byte

플래시 메모리는 지움 연산 횟수의 제한이 있고, 제자리 업데이트 시 지움 연산이 선행되어야 한다[1].

NAND 플래시의 읽기, 쓰기 단위는 페이지(512B)이며 블록은 32개의 페이지로 구성된다. 지움 연산은 블록단위로 작동하며 <표 1>에서와 같이 쓰기 연산에 비해 지움 연산이 상당히 많은 시간이 소요된다. YAFFS의 경우 페이지 단위로 쓰기 연산에 앞서 쓰기 공간 확보를 하는 가비지 컬렉션(Garbage Collection)을 수행하며, 지움 대상 블록이 많은 경우에 쓰기 지연의 원인이 된다. 가비지 컬렉션 이란 유효한 데이터가 가장 적은 블록을 선택하여 유효한 데이터를 새로운 영역에 복사하고 지움 연산을 수행하는 일련의 과정을 말한다.

이러한 쓰기 연산 지연을 예방하기 위하여 본 논문에서는 지움 연산 횟수를 특정 시간 간격 동안 제어하여 쓰기 연산의 지연을 보완하고, 쓰기 연산이 적게 일어 나는 시기에 지움 연산 시도 횟수를 증가시켜 여유 공간을 확보 하는 적응적인 가비지 컬렉션 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2 장에서는 YAFFS 에 대한 소개, 3 장에서는 적응적인 가비지 컬렉션 기법의 설계 및 구현, 4 장에서는 실험 및 성능평가, 5 장에서는 결론 및 향후 과제에 대하여 분석한다.

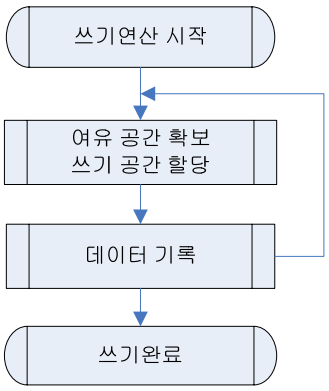
이 논문은 2 단계 두뇌한국21 사업에 의하여 지원되었음.

2. 관련연구

2.1 YAFFS

대표적인 NAND 플래시 파일 시스템인 YAFFS[5]는 페이지(page) 단위로 읽기, 쓰기 연산을 수행하며 마운트 시 플래시 메모리의 모든 영역을 읽어 각 블록 상태정보와 Inode 정보, Tnode 정보를 램(Ram)에 유지한다. 블록 상태정보는 사용하고 있는 페이지 수(pagesInUse), 블록상태(blockState) 로 구성되며, Inode 정보는 디렉터리간의 관계와 파일 정보로 구성된다. Tnode는 데이터의 물리적인 위치 정보를 저장하고 있다. 지움 연산 대상 블록을 찾기 위해 메모리에 있는 블록 상태정보를 읽어, 지움 대상 블록의 유효한 데이터를 복사 후 해당 파일 Inode 정보, Tnode 정보를 갱신한다. 또한, YAFFS는 쓰기 연산 시 지움 연산을 선행하여 쓰기 공간을 확보한다. 이는 지움 대상 블록이 많은 경우 쓰기 지연의 원인이 된다.

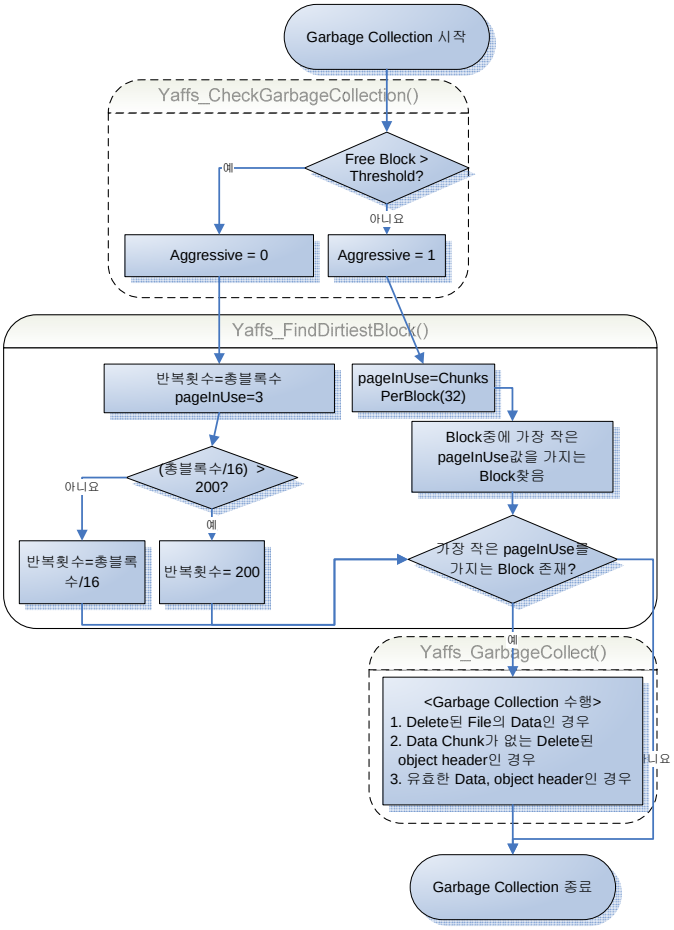
YAFFS의 쓰기 연산과정은 (그림 1) 과 같이 쓰기 연산이 시작되면 지움 대상 블록을 찾아 쓰기 공간 확보를 한다. 지움 대상 블록이 있을 시 지움 연산을 수행하고 연산이 완료되면 플래시 메모리에 쓰기 영역을 할당 받아 데이터를 기록한다. 이와 같은 쓰기 과정은 페이지 단위로 반복적으로 수행된다.



(그림 1) 쓰기연산 과정

쓰기 공간 확보 과정은 파일 시스템 작동을 위한 최소의 블록수인 예약블록(reserved block) 수에 따라 두 가지 경우로 수행된다. 예약블록이 충분히 확보되어 있는 경우에는 (그림 2) 와 같이 aggressive의 값이 0이 되어 블록 내 유효 페이지(valid page) 수가 3 이하인 블록들 중에 가장 작은 유효 페이지를 가지는 블록을 찾아 지움 연산을 수행한다. 지움 대상 블록을 찾기 위해 검색하는 블록 수는 200 개로 제한된다. 예약 블록이 충분히 확보되어 있지 않은 경우에는 aggressive 의 값이 1이 되어 플래시 메모리내의 전체 블록을 대상으로 유효 페이지수가 가장 작은 블록을 지움 대상으로 선택하고 지움 연산을 수행한다.

(그림 1) 의 쓰기 연산 과정과 같이 매번 쓰기 연산이 발생할 때 지움 연산을 수행하게 되면 임베디드 시스템의 성능에 영향을 미칠 수 있다. 실시간 운영체제(real-time operating system)의 경우 YAFFS의 쓰기 연산과정은 지움 연산 때문에 임계시간(threshold time)을 초과 할 가능성이 있다. 따라서, 본 논문에서는 YAFFS의 지움 연산을 읽기, 쓰기 연산량에 따라 제어하고, 연산이 작게 일어 나는 경우에 더 많은 지움 연산 기회를 부여하여 시스템의부하를 조절하는 효율적인 가비지 컬렉션 기법을 제안한다.



(그림 2) 지움 연산 수행 과정

3. 설계 및 구현

3.1 적응적인 가비지 컬렉션 기법 설계

제안하는 가비지 컬렉션 기법은 읽기, 쓰기 연산량을 측정하여 지움 연산 횟수를 제어한다. 이는 쓰기 연산의 누적 평균을 측정하여 지움 연산 횟수를 제어한다. 쓰기, 지움 연산량이 감소하면 지움 연산 시도 횟수를

증가시키고, 쓰기 연산량이 증가하면 지움 연산 시도 횟수를 감소시켜 지움 연산으로 인한 시스템의 부하를 조절하게 된다. 쓰기 연산량과 지움 연산량이 적은 경우에는 연산량을 측정하는 시간 간격을 짧게 하여 연산량이 증가 하는 경우를 감지하게 한다. 연산량이 증가 하는 것을 감지하면 지움 연산 시도 횟수를 감소시켜 시스템 부하를 조절하게 된다.

파일 시스템 마운트 후 쓰기 연산의 n 번째 시간까지의 누적 평균 E_n 을 (식 1) 과 같이 표현할 수 있다.

$$E_n = \frac{(1-\alpha)\times\sum_{i=1}^{n-1} W_i}{n-1} + (\alpha \times W_{cur}) \qquad \text{(식 1)}$$

(식 1) 은 기존의 가비지 컬렉션 평준화 기법[4]으로 α 는 가중치(0<α<1)를 나타내며 W_i 은 i 번째 시간 간격 동안 쓰기 요구량을 나타내고, W_{cur} 는 가장 최근 시간 간격 동안의 쓰기 요구량을 나타낸다.

가비지 컬렉션 평준화 기법은 쓰기량에 따라 지움 대상 블록에 대한 검색 시도 횟수를 결정하여 쓰기 연산을 보장한다. 가장 최근의 시간 간격 동안의 쓰기 요구량을 따로 구분한 이유는 가장 최근 시간 간격 동안에 쓰기 요구량(W_{cur})이 많으면 다음 시간 간격 동안에도 쓰기량이 많을 가능성이 높기 때문에 이를 고려한 것이다. 가비지 컬렉션 평준화 기법은 연산량이 적은 경우에도 지움 연산의 횟수가 쓰기 연산량에 의존적이어서 비효율적으로 작동한다. 따라서, (식 1) 을 쓰기, 지움 연산량에 따라 적응적으로 가비지 컬렉션이 일어 나도록 하기 위하여 (식 2)와 같이 표현할 수 있다.

$$E_{adj} = E_n \times (1 \pm \beta) \times T_{interval}, (0 < T_{interval}, \beta < 1)$$

(식 2)

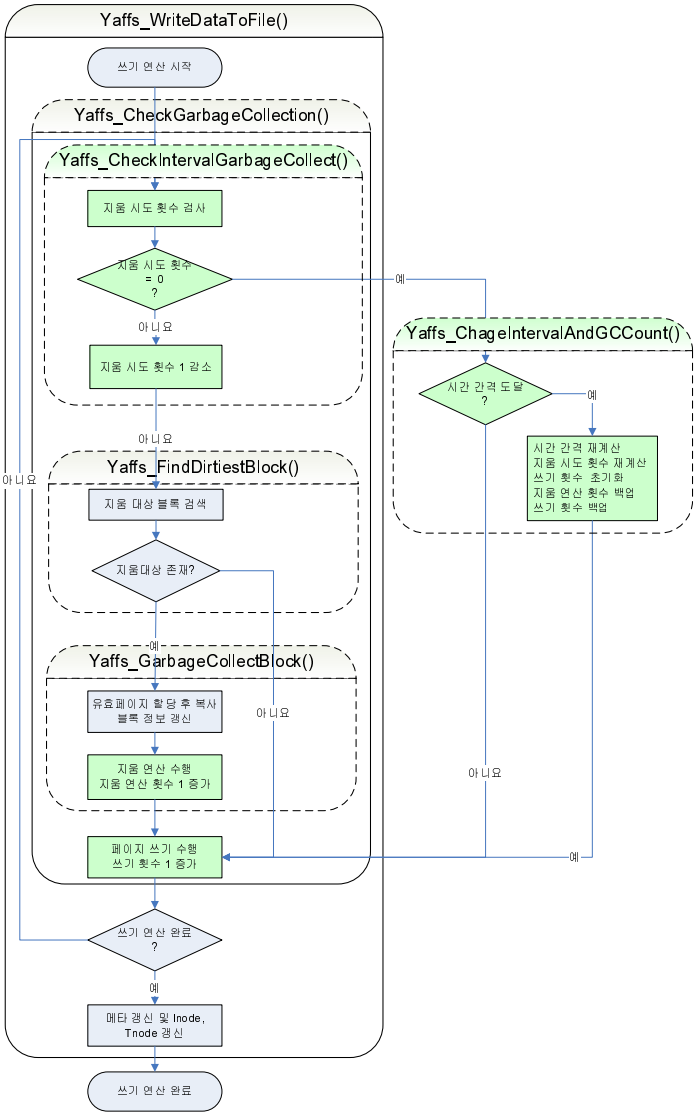
E_{adj} 는 다음 시간 간격 동안의 지움 시도 횟수, β 는 지움 연산 증가 비율, $T_{interval}$ 는 시간 간격을 의미하며 연산량에 따라 변한다.

(식 2)는 기존의 가비지 컬렉션 평준화 기법에 시스템 연산량 변화에 따라 쓰기 공간확보를 더욱 용이하도록 개선한 것으로 적응적인 가비지 컬렉션 기법 구현 시 이용할 공식이다.

3.2 적응적인 가비지 컬렉션 기법 구현

제안하는 가비지 컬렉션 기법을 YAFFS에 구현하기 위해서 가비지 컬렉션 평준화 기법[4]를 (그림 3) 와 같이 수정 하였다.

기존의 YAFFS에 연산량 측정을 위해 임의로 정한 시간 간격 동안의 쓰기 수, 지움 연산 횟수를 의미하는 변수를 추가하였다. Yaffs_CheckIntervalGarbage-Collect() 는 시간 간격 내에 허용된 지움 시도 횟수를 모두 소비 하였는지 여부와 시간 간격에 도달 여부에 따라 Yaffs_ChangeIntervalAndGCCount() 함수를 호출하여 쓰기 수, 지움 시도 횟수를 초기화 된다.



(그림 3) 적응적인 가비지 컬렉션 작동 과정

또한, yaffs_ChangeIntervalAndGCCount() 함수는 (식 2) 의 지움 연산 증가 비율 β 에 따라 지움 시도 횟수를 증가 혹은 감소 시킨다. 지움 시도 횟수가 모두 소비되지 않았다면, 지움 대상 블록을 검색하는 Yaffs_Find-DirtiestBlock() 함수를 호출 하여 지움 연산을 수행한다. 지움 연산이 완료 되면 쓰기 연산을 수행하고, 시간 간격 동안 쓰기 연산 횟수를 의미하는 쓰기 연산 수를 1 증가 시킨다. Yaffs_WriteDataToFile() 함수에 의한 쓰기 연산이 완료 되면 파일 오브젝트 헤드 정보 (Inode) 를 갱신 하고, 데이터의 물리적인

위치를 나타내는 Tnode를 갱신한다.

적응적인 가비지 컬렉션 기법을 YAFFS에 구현 하기 위해 <표 2> 와 같은 함수를 추가하였다.

<표 2> 적응적인 가비지 컬렉션 기법의 주요함수

함 수 명	기 능
GetAvgGCCount()	지움 연산 평균수를 구하는 함수
GetAvgWriteCount()	쓰기 연산 평균수를 구하는 함수
LastIntervalGCCOunt()	가장 최근 시간 간격 동안 지움 연산 횟수를 구하는 함수
LastIntervalWriteCount()	가장 최근 시간 간격 동안 쓰기 수를 구하는 함수
QueryCheckUpBusy()	연산량을 계산하는 함수
UpdateRequestSpaceSize_Avg2()	(식 2) 를 구현한 함수
ChangeIntervalAndGCCount()	연산량이 작은 경우에 시간 간격과 지움 연산 횟수를 변화 시키는 함수
ResetInterval()	시간 간격을 초기화 하는 함수

구현상에 가장 큰 문제점은 실수 연산이 지원 되지 않는 것이었으며, 소수점 자리 연산을 피하기 위해 곱하기 연산 후 나누기 연산을 수행 하였다.

4. 실험 및 성능평가

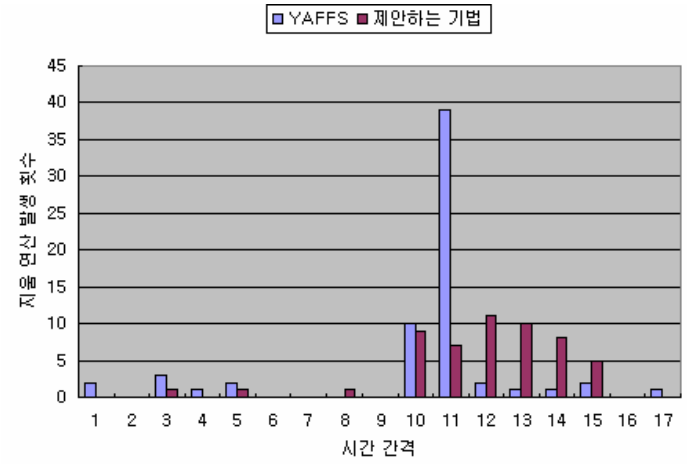
실험 환경은 <표 3> 와 같이 PXA255-Pro III 임베디드 실험용 보드에서 하였다. 64MB NAND 플래시 메모리에 수정한 YAFFS 파일 시스템을 마운트 하여 제안한 적응적인 가비지 컬렉션 기법의 성능 평가를 실험 하였다.

읽기, 쓰기, 지움 연산량 비율은 트레이스 센터(Trace Distribution Center)에서 제공하는 트레이스 데이터를 분석 하여 적용하였다[6].

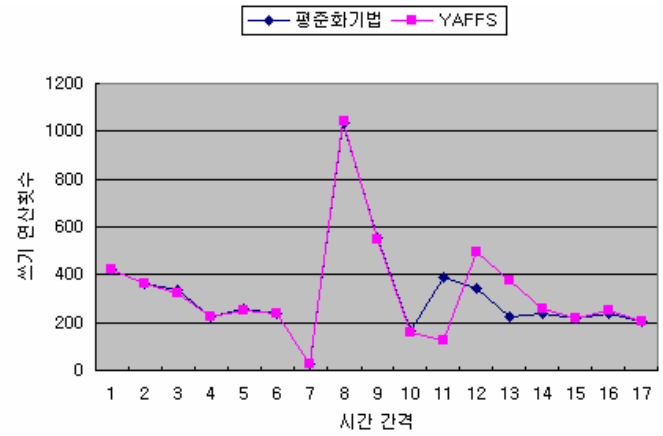
<표 3> PXA255-Pro III 임베디드 보드

항 목		명 세
CPU		Intel PXA255
SDRAM		128 MB SDRAM
Flash Memory	NOR	32 MB (Intel)
	NAND	64 MB (Samsung)

실험방법은 제안하는 기법을 적용한 YAFFS와 기존의 YAFFS에 동일한 연산량을 적용하였고, $T_{interval}$ 을 1000ms 으로 하였고, 쓰기와 지움 연산량이 누적 평균의 50%에 못 미칠 경우에는 $T_{interval}$ 을 200ms 이 되도록 하였다. 지움 시도 횟수는 누적 평균의 50% 비율로 증가 시켜 실험하였다.



(그림 3) YAFFS와 제안하는 알고리즘의 특정 시간 간격 동안에 발생한 지움 횟수 비교



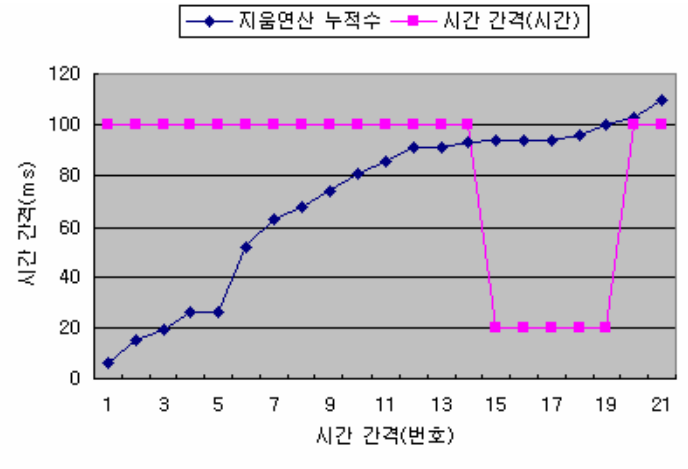
(그림 4) YAFFS와 제안하는 알고리즘의 특정 시간 간격 동안에 발생한 쓰기 연산 횟수 비교

(그림 3)와 같이 YAFFS 의 가비지 컬렉션의 경우 11번째 시간에서 볼 수 있듯이 모든 지움 대상 블록에 대해 지움 연산을 수행하였고, 지움 연산에 의해 쓰기 연산이 지연되는 것을 보여준다. (그림 4) 에서는 제안하는 기법은 11번째 시간과 15번째 시간 사이에서 지움 연산 횟수를 제한하여 쓰기 연산에 용이하게 작동하는 것을 볼 수 있다.

(그림 5) 와 (그림 6) 는 제안하는 알고리즘의 시간 간격의 변화와 지움 연산의 시도 횟수를 보여준다. 실험을 위해 쓰기 연산과 지움 연산이 작은 경우에서 실험 하였다. 15번째와 19번째 시간 간격 사이에서

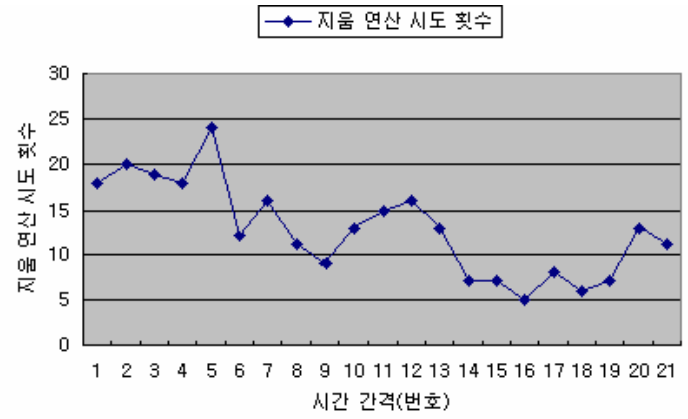
$T_{interval}$ 이 200 ms 으로 변화 된 것을 볼 수 있다.

(그림 5) 에서 14번째 시간 간격 동안 연산수가 적게 발생 하였기 때문에 시간 간격을 줄이고 지움 연산 시도 횟수를 늘려 주었다. 연산수가 적어 부하가 작은 때 지움 연산을 하여 여유 공간을 확보하는 것이다.



(그림 5) 제안하는 알고리즘의 시간 간격 변화

그리고, (그림 6) 에서 15번째와 19번째 시간 간격 사이에서 지움 연산 횟수가 줄어 든 것 처럼 보이지만 시간 간격이 100 에서 20으로 변화 된 것을 감안하면 시간 간격 100 동안 실제로는 33회 지움 연산 시도를 하였다.



(그림 6) 제안하는 알고리즘의 지움 연산 시도 횟수

따라서, 실험 결과 제안하는 적응적인 가비지 컬렉션 기법은 쓰기 연산의 지연을 예방하고 연산이 적게 발생하는 경우에 지움 연산 시도를 늘려 쓰기 공간 확보에 용이 하다는 것을 알 수 있다.

5. 결론 및 향후 과제

제안한 기법을 YAFFS에 적용한 실험을 통해 연산이 작게 발생하는 경우에 지움 연산 시도횟수가 늘어나 쓰기 공간확보에 용이하게 작동하는 것을 보았고, 가비지 컬렉션 횟수가 평준화 되어 읽기, 쓰기 지연시간이 줄어든 것을 보았다. 하지만, 지움 연산 증가 비율과 시간 간격 결정을 위한 다양한 환경에서의 실험이 추가적으로 필요할 것이다.

참고문헌

[1] Intel,"Understanding the Flash Translation Layer (FTL) specification".

[2] White Paper: Two Technologies Compared: NOR vs. NAND".

[3] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. K. Cho, "A Space-efficient Flash Translation Layer for CompactFlash Systems," IEE Transactions on Consumer Electronics, Vol. 48, No. 2., pp. 366-375, May 2002

[4] 이상기, 이태훈, 정기동, "임베디드 시스템을 위한 가비지 컬렉션 평준화 기법", 한국정보처리 학회 논문집 제 13권 제 2호

[5] D. Woodhouse, Red Hat,inc. "JFFS: The journaling Flash File System."

[6] Li-Pin Chang and Tei-Wei Kuo, "An Adaptive Stripping Architecture for Flash Memory Storage Systems of Embedded Systems," IEEE Eighth Real-Time and Embedded Technology and Applications Symposium (RTAS), San Jose, USA, Sept 2002.