F2FS와 Ext4의 FTL별 성능 비교 및 BAST Padding 기법 제안

최시훈○ 신동군
성균관대학교 정보통신대학
beswan08@gmail.com, dongkun@skku.edu

Performance Comparison between EXT4 and F2FS under Different FTL Algorithms

Sihoon Choi○, Dongkun Shin
College of Information and Communication Engineering,
Sungkyunkwan University.

요 약

본 연구에서는 플래시 메모리를 사용하는 환경에서 각 파일 시스템별 FTL간의 성능을 측정하여 상호 비교 분석하였으며, F2FS의 동작에 적합한 FTL로서 BAST에 Padding 기법을 적용한 FTL을 제안하였다.

1. 서 론

플래시 메모리는 낮은 전력 소모와 충격에 강하고 소형화가 가능하다는 특징을 가지고 있어 스마트폰을 비롯한 모바일 기기의 저장장치로 활용되고 있다.

플래시 메모리는 여러 개의 블록으로 구성되어 있으며 각 블록은 여러 개의 페이지로 구성되어 있다. 페이지는 읽기 및 쓰기 작업의 단위가 되며, 페이지에 쓰기 전에 해당 페이지는 지워진 상태여야 한다. 그러나 지우기는 블록으로 이루어지기 때문에 쓰고자 하는 페이지와 같은 블록에 속한 다른 유효한 페이지의 복사가 필요하다. 이러한 특징을 다루기 위해서 FTL(Flash Translation Layer)이라는 소프트웨어 계층이 존재한다.[1]

플래시 메모리의 성능은 이를 관리하는 FTL에 따라 서로 다른 양상을 보인다. 매핑의 단위가 작으면 유연한 관리가 가능하지만 맵의 크기가 비대해져 많은 RAM을 요구한다. 반대로 매핑의 단위가 크면 맵의 크기가 작은 반면에 물리 공간의 관리를 위한 오버헤드가 크다. 그러나 FTL은 공통적으로 연속적인 쓰기 동작 패턴에 대해 더 빠른 처리가 가능하다. 플래시 메모리의 병렬성을 최대한 활용할 수 있으며, 더 적은 량의 지우기 연산을 발생시킨다. 이러한 특징에 기인하여 플래시 메모리의 성능을 최대한 활용할 수 있도록 플래시 메모리용 파일 시스템인 F2FS(Flash-Friendly File System)[2]가 개발되었다.

본 연구는 F2FS와 Ext4 파일 시스템이 플래시 메모리와 각 FTL 상에서 어떤 성능 양상을 보이는지 관찰하였으며, BAST의 로그 블록을 할당받기 위한 낭비를 줄일 수 있는 Padding 기법을 제안하였다.

2. 관련 연구

2.1. BAST(Block Associative Sector Translation)[3]

BAST는 블록 매핑과 페이지 매핑을 함께 사용하는 매핑 기법이다. 로그 블록은 페이지 매[1]핑을 사용하고, 데이터 블록은 블록 매핑으로 관리한다. 맵 크기가 작기 때문에 RAM에 모든 맵 데이터를 저장하여 맵 로딩으로 인한 오버헤드가 없다. 그러나 로그 블록은 한정적이며, 하나의 데이터 블록에 대한 로그 버퍼로 하나의 로그 블록만을 할당하기 때문에 쓰기 요청의 집약성이 떨어질 경우 로그 블록의 지우기 동작이 많아진다.

2.2. DFTL[4]

DFTL은 RAM 크기가 맵보다 작은 경우를 위한 페이지 매핑으로, 맵 데이터를 플래시 메모리에 저장하고 접근이 필요한 주소의 맵을 RAM으로 불러와서 처리한다. IO요청의 집약성이 높은 경우 페이지 매핑과 동일한 성능을 보이지만, 그렇지 않은 경우 잦은 플래시 메모리에 저장된 맵 접근에 따른 오버헤드가 발생한다.

2.3. F2FS (Flash-Friendly File System)

F2FS는 LFS(Log-structured File System)의 일종으로 FTL을 통해 동작하는 플래시 메모리에 적합하게 개발된 파일 시스템이다. Hot/Cold 데이터의 구분으로 효율적인 클리닝이 가능하며, 높은 공간 사용량에 대비하여 오버헤드를 발생시키는 클리닝의 양을 줄이고자 SSR이라는 기법을 함께 사용한다. SSR은 클리닝 되지 않은 Invalid된 공간에 새로운 데이터를 덮어쓰는 기법이다.

2.4. BPLRU[5]

BPLRU는 버퍼를 관리하는 기법으로, 블록 단위로 LRU 리스트를 관리하며 버퍼에서 내보낼 때 블록 전체를 in-place 방식으로 패딩해서 플래시 메모리에 기록한다. 추가적인 복사 연

* 이 논문은 2013년도 정부(교육부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(2013R1A1A2A10013598)

산이 오버헤드로 작용하지만 GC(Garbage Collection)로 인한 오버헤드가 감소하는 장점이 있다.

3. BAST Padding 구현

쓰기 요청된 페이지에 대해 그 이전에 기록된 데이터 블록의 모든 페이지를 로그 블록에 in-place로 패딩하여 복사하는 연산을 추가하였다. [그림 1]에서 8개 페이지로 구성되어 있는 데이터 블록의 모든 데이터가 유효한 상태에서 5번째 페이지에 대한 업데이트가 요청될 경우 해당 페이지 이전의 모든 페이지의 내용을 로그 블록으로 복사한다. 그러나 이후에 패딩된 페이지에 대한 업데이트가 다시 발생할 경우 패딩을 포기한다.

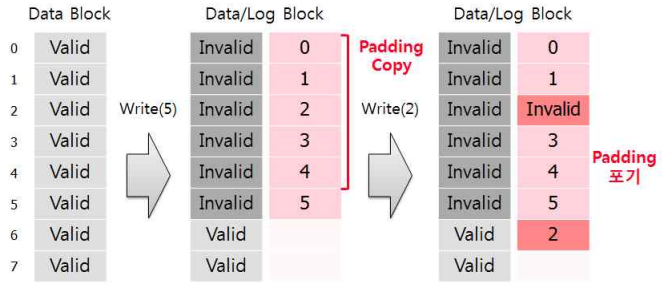

그림 1. BAST Padding의 동작

4. 실험 환경

실험은 Intel(R) Core^{tm} i7-3770 3.40GHz 프로세서, 8GB DRAM을 장착한 서버를 사용하였으며, Ubuntu 12.04LTS 64bit(Linux Kernel 3.4.7) 운영체제에서 이루어졌다. 사용한 실험 보드는 OpenSSD를(ARM7TDMI-S 87.5Mhz 프로세서, 96KB SRAM, 64MB DRAM, S-ATA 2.0)[6] 사용하였다. OpenSSD 보드에는 4개의 삼성 K9LCG08U1M 32GB 낸드 플래시 칩으로 구성된 플래시 모듈 두 개를 장착하였으며, 실험에서는 4GB만을 사용할 수 있도록 제한하였다.

OpenSSD에 사용한 FTL은 BAST와 이에 패딩 기법을 적용한 BAST Padding과 Full Page이다. BAST와 BAST Padding은 Superblock[7] 기법을 적용하여 4개 물리 블록 및 페이지가 하나의 가상 블록으로 작동한다. 각 FTL의 상세 사항은 표 1.과 같다.

표 1. 실험에 사용한 FTL 상세 사항

	BAST(Padding)	Full Page
크기	4GB	
블록당 페이지 수	128개	
페이지 크기	32KB	8KB
로그 블록 수	64개 (256MB)	-
오버프로비젼	-	340MB

마지막으로 벤치마크에 사용한 툴은 IOZone 3.397 Version[8]이다. 실험 시나리오는 실험 전 초기 공간 사용량을 조절하고 순차 쓰기로 파일을 생성하고, 실제 사용 환경에서 파일 일부에 대한 업데이트가 존재할 것을 상정하여 파일의 25%에 대해 임의 업데이트를 한 후에 실험을 진행하였다. 실험은 순차 읽기, 임의 읽기/쓰기, 순차 쓰기 순서로 수행하였다. 실험 변수는 표 2.와 같이 설정하였다.

표 2. 실험 변수

IO 요청 크기	4KB
Utilization	20%, 50%, 80%, 90%, 95%
테스트 파일 크기	600MB
F2FS Overprovision	250MB
F2FS Reserved	50MB

5. 실험 결과

5.1 읽기 성능

[그림 2]과 [그림 3]의 그래프에서 읽기 성능은 파일 시스템 간 차이는 크지 않았다. 파일시스템의 메타 데이터를 읽는 시간은 플래시 메모리 접근 시간에 비해 매우 적기 때문이다.

IO스케쥴러에서 순차 읽기 명령들은 1MB 크기로 병합된다. 그러나 OpenSSD는 한 번의 플래시 접근에 한 개의 페이지만을 읽기 때문에, 페이지의 크기가 작은 페이지 매핑의 [그림 2]의 순차 읽기 성능이 떨어지는 것으로 보인다.

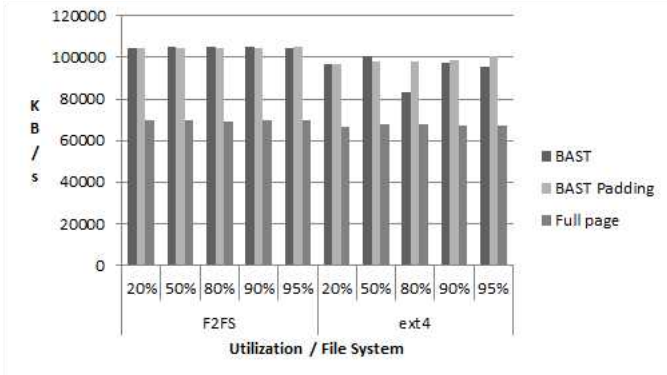

그림 2. 순차 읽기 성능

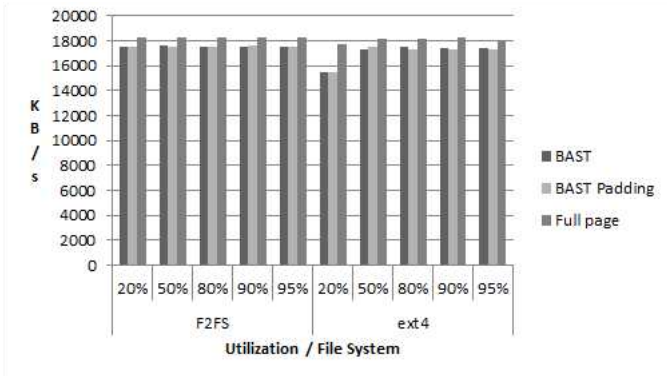

그림 3. 임의 읽기 성능

5.2. 쓰기 성능

페이지 매핑에서 대부분의 조건 하에서 임의 쓰기와 순차 쓰기의 성능이 비슷한 결과를 보인다. FTL 내부에서 물리 페이지를 순차적으로 사용하고 오버프로비젼으로 인해 FTL의 GC를

늦추기 때문이다. 그러나 F2FS에서 GC가 일어나 쓰기 요청이 크게 증가하는 사용량 95%에서 성능이 급감하는 것을 볼 수 있다.

전반적으로 페이지 매핑의 성능이 낮은 것은 지우기 연산에 따른 오버헤드가 원인이다. 페이지 매핑의 지우기 연산 횟수가 많은데 이는 BAST와 BAST Padding이 한 번의 지우기 연산에 블록 4개를 지우는데 반해 페이지 매핑은 블록 1개만을 지우기 때문이다.

BAST보다 BAST Padding의 성능이 뛰어난 원인은 로그 블록이 in-place로 기록되어 있어 낮은 비용으로 지우기 연산을 수행할 수 있으며 빈 페이지를 많이 얻을 수 있어 지우기 연산이 적게 필요하다. Ext4의 임의 쓰기 패턴의 경우 패딩이 실패할 가능성이 높으며, 패딩 오버헤드가 큰 반면 F2FS를 사용하는 환경에서 BAST 대비 뛰어난 성능을 보였다.

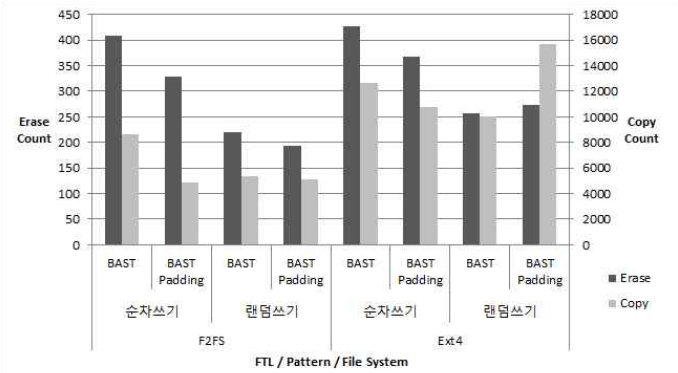

그림 4. 사용량 20%일 때
BAST와 BAST Padding의 오버헤드 비교

순차 쓰기 실험에서 Ext4에 비해 F2FS의 성능이 뛰어난 원인은 실험 파일이 임의 쓰기에 의해 흩어져 있어 Ext4의 경우 25%의 쓰기 요청이 임의한 패턴을 가지는데, F2FS는 모든 쓰기 요청이 순차적인 패턴을 가지기 때문이다.

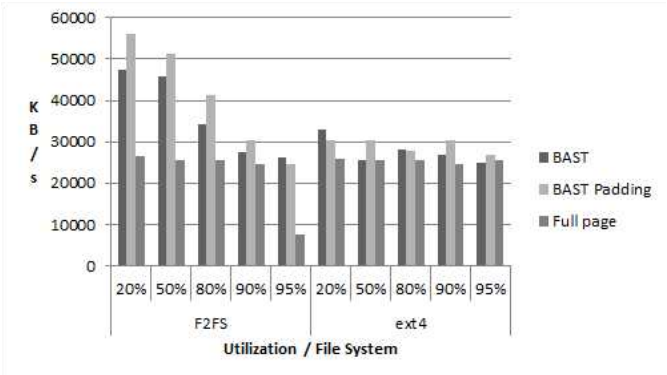

그림 5. 순차 쓰기 성능

임의 쓰기 성능은 로그 방식으로 순차적으로 기록하는 F2FS가 높다. F2FS에 BAST, BAST Padding을 사용한 경우 할당된 로그 블록의 활용률이 높아 지우기 연산이 적으며 적은 비용의 지우기가 가능하기 때문이다.

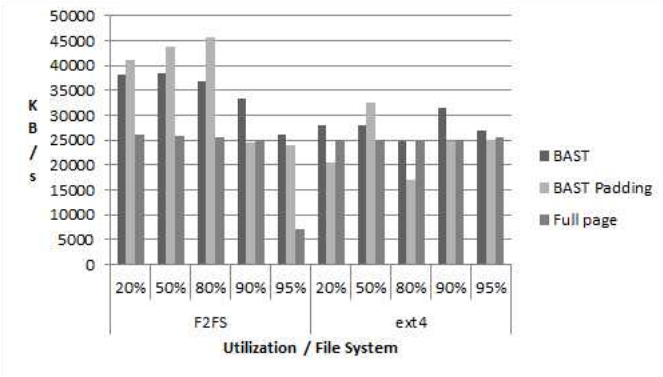

그림 6. 임의 쓰기 성능

5. 결론 및 향후 연구

본 연구에서는 F2FS와 Ext4 파일 시스템에 대해 각 FTL에서의 성능을 측정하여 F2FS가 플래시 메모리에 적합한 파일 시스템임을 확인하였고, F2FS에서 BAST에 Padding 기법을 적용한 FTL을 제안하여 추가적인 성능 향상을 얻을 수 있음을 확인하였다.

향후 연구 과제로 BAST Padding이 실패 시 패딩을 포기하지 않고 새로운 로그 블록을 할당받는 기법과 패딩 오버헤드를 낮추기 위해 선택적으로 패딩하는 기법을 구현하여 실험 할 계획이다.

6. 참고 문헌

[1] Ban, Amir., “Flash file system.“, U.S. Patent No. 5,404,485. 4 Apr. 1995.

[2] Flash Memory Filesystem, Korea Linux Forum, Oct. 12, 2012, http://elinux.org/images/8/81/A_New_File_System_Designed_for_Flash_Storage_in_Mobile.pdf

[3] Kim, Jesung, et al., “A space-efficient flash translation layer for compactflash systems.“, Consumer Electronics, IEEE Transactions on 48.2 (2002): 366-375.

[4] Gupta, Aayush, Youngjae Kim, and Bhuvan Urgaonkar., DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings., Vol. 44. No. 3. ACM, 2009.

[5] Kim, Hyojun, and Seongjun Ahn., “BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage.“, FAST. Vol. 8. 2008.

[6] OpenSSD project, http://www.openssd-project.org

[7] Kang, Jeong-Uk, et al., “A superblock-based flash translation layer for NAND flash memory.“, Proceedings of the 6th ACM & IEEE International conference on Embedded software. ACM, 2006.

[8] Norcott, William D., and Don Capps., “Iozone filesystem benchmark.“ URL: www.iozone.org 55 (2006).