# 참조 패턴에 따라 페이지 및 블록 사상 영역의 크기를 조절하는 Janus-FTL

## (Janus-FTL Adjusting the Size of Page and Block Mapping Areas using Reference Pattern)

권 훈 기 † 김 은 삼 ††
(Hunki Kwon) (Eunsam Kim)
최 종 무 ††† 이 동 희 ††††
(Jongmoo Choi) (Donghee Lee)
노 삼 혁 †††††
(Sam H. Noh)

**요 약** 본 논문에서는 참조 패턴에 따라 페이지 사상 정책과 블록 사상 정책을 선택적으로 사용하는 Janus-FTL을 제안한다. 일반적으로 플래시 메모리의 특성에 따르면, 순차 참조의 경우 블록 사상 FTL이 적당하고, 비 순차적인 참조의 경우 페이지 사상 FTL이 적당하다. 따라서 실용적인 FTL은 데이터의 특성에 따라 플래시 메모리 블록을 블록 사상 또는 페이지 사상 정책으로 선택적으로 사용하면서, 블록 사상 영역과 페이지 사상 영역의 크기를 참조 패턴에 따라 효율적으로 변화하여 할당하는 관리 기법이 필요하다. 본 논문에서는 저장된 데이터가 블록 사상 영역에서 페이지 사상 영역으로 이동하는 퓨전(Fusion) 연산과 반대로 이동하는 디퓨전(Defusion) 연산을 통해, 블록 사상과 페이지 사상 기법을 동시에 활용하는 Janus-FTL을 설명한다. 또한 Janus-FTL을 구현하여 성능을 측정하였으며, 성능 측정 결과에 따르면 기존의 FTL에 비해 최대 50%의 우수한 성능을 보였다.

**키워드** : 플래시 메모리 저장장치, 페이지 사상, 블록 사상, 쓰레기 수집, FTL(Flash-memory Translation Layer)

***Abstract*** Naturally, block mapping FTL works well for sequential writes while page mapping FTL does well for random writes. To exploit their advantages, a practical FTL should be able to selectively apply a suitable scheme between page and block mappings for each write pattern. To meet that requirement, we propose a hybrid mapping FTL, which we call Janus-FTL, that distributes data to either block or page mapping areas. Also, we propose the fusion operation to relocate the data from block mapping area to page mapping area and the defusion operation to relocate the data from page mapping area to block mapping area. And experimental results of Janus-FTL show performance improvement of maximum 50% than other hybrid mapping FTLs.

**Key words** : Flash memory storage, Page mapping, Block mapping, Garbage collection, FTL (Flash-memory Translation Layer)

## 1. 서 론

NAND 플래시 메모리는 무게가 가볍고 충격에 강하며 전력 소모가 적어 휴대 전화나 PDA와 같은 이동 기기의 저장장치로 널리 사용되고 있다. 그리고 최근에는 SSD(Solid State Drive) 형태로 노트북 컴퓨터와 고성능 서버까지 그 적용 범위가 확대되고 있다. 그런데 플래시 메모리는 블록이라는 비교적 큰 단위로 소거한 후 페이지라는 작은 단위로 데이터를 읽거나 써야 하며, 각 블록마다 소거 횟수에 한계가 있는 것과 같은 다양한 제약이 존재한다. 결국 이러한 제약을 극복하기 위하여 플래시 메모리 저장장치는 논리 섹터 번호를 물리적인 위치로 사상하는 FTL(Flash-memory Translation Layer)이라는 소프트웨어 계층을 사용하며, FTL을 통해 기존의 저장장치와 같이 플래시 메모리에 섹터 단위로 읽기/쓰기가 가능하도록 한다[1].

NAND 플래시 메모리 FTL들은 사상 단위에 따라 블록 사상 FTL과 페이지 사상 FTL로 분류할 수 있다. 페

· 본 연구는 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업(No. R0A-2007-000-20071-0)과 2008년 정부(교육과학기술부)의 재원으로 한국학술진흥재단(KRF-2008-314-D00340)의 지원을 받아 수행된 연구임.
· 이 논문은 2009 한국컴퓨터종합학술대회에서 '참조 패턴에 따라 페이지 사상 영역과 블록 사상 영역의 크기를 조절하는 Janus-FTL'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서울시립대학교 컴퓨터과학부
kwonhunki@uos.ac.kr
†† 정 회 원 : 홍익대학교 정보컴퓨터공학부 교수
eskim@hongik.ac.kr
††† 종신회원 : 단국대학교 컴퓨터학과 교수
choijm@dankook.ac.kr
†††† 정 회 원 : 서울시립대학교 컴퓨터과학부 교수
dhl_express@uos.ac.kr
††††† 종신회원 : 홍익대학교 정보컴퓨터공학부 교수
samhnoh@hongik.ac.kr

논문접수 : 2009년 8월 17일
심사완료 : 2009년 10월 5일

Copyright©2009 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.
정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제12호(2009.12)

이지 사상 FTL은 섹터 단위 또는 플래시 메모리의 읽기/쓰기 단위인 페이지 단위로 사상하며 그 동작 방식은 LFS (Log-structured File System)[2]과 유사하다. 페이지 사상 기법의 경우, 블록 사상 기법과 비교하여, 임의(비 순차적인) 쓰기 성능을 향상 시킬 수 있지만, 사상 정보를 가지는 맵(Map)의 크기가 매우 크며, 쓰레기 수집과 같은 블록 재활용 기법의 효율에 따라 성능 변화가 심하다.

대용량 NAND 플래시 메모리를 사용하는 메모리 카드용 FTL 또는 임베디드 시스템용 FTL(예, M-Systems의 NFTL)에서는 주로 블록 단위 사상 기법이 이용된다. 블록 단위 사상의 경우 맵의 크기가 작고, FTL의 동작 방식이 단순하여 저 사양의 임베디드 시스템에서도 효율적으로 구현되며, 필요한 경우 하드웨어/소프트웨어 통합 설계를 통하여 성능 향상을 가질 수 있다는 장점이 있다. 하지만 블록 사상 FTL은 순차 쓰기에 최적화 되어 있지만 임의 쓰기에는 성능이 떨어지는 단점을 가진다.

이와 같이 블록 사상 기법은 순차 쓰기 성능이 우수하지만 지역성을 보이는 참조나 임의 쓰기는 효율적으로 처리하지 못한다. 반면 페이지 사상 기법의 경우 지역성을 보이는 참조나 임의 쓰기 성능이 블록 사상 기법보다 우수하다. 최근에는 두 가지 사상 기법의 특성을 활용하는 하이브리드(Hybrid) 사상 기법이 제안되었다[3-5]. 이러한 하이브리드 사상 기법은 데이터 영역은 블록 사상 기법으로 관리하고 로그 영역은 페이지 사상 기법으로 관리하여 블록 사상 기법에서 자주 발생하는 전체 병합(Full merge)을 줄이는데 목적이 있다.

저장 장치의 참조 패턴을 보면 순차와 비 순차적인 참조들이 뒤섞여 있으며, 일부 섹터들은 지역성을 보이면서 재 참조되는 경향이 있다. 만약 FTL이 이러한 참조 패턴에 최적화 되어, 순차 쓰기 요청은 블록 사상 기법을 사용하는 블록에 순차적으로 저장하고, 일부 지역성을 보이는 비 순차적인 쓰기 요청의 경우 페이지 사상 기법으로 관리한다면 몇 가지 이득을 얻을 수 있다. 먼저 순차 참조되는 데이터들은 블록 사상 영역에 저장하여 효율적으로 병합 연산을 수행하면서, 지역성을 보이는 일부 데이터에 대해 페이지 사상 기법을 적용하여 전체적인 성능을 향상시킬 수 있다. 아울러 일부 데이터에 대해서만 페이지 사상 기법을 적용하므로, 상대적으로 적은 양의 여유 공간이 주어질 때 낮은 이용율로 인해 페이지 사상 영역의 쓰레기 수집을 효율적으로 수행할 수 있다. 만약 페이지 사상 영역의 데이터가 참조되지 않으면 이 데이터가 포함된 블록을 다시 블록 사상 영역으로 환원하고, 자주 참조되는 블록 사상 영역의 데이터를 페이지 사상 영역에 포함되도록 하면, 참조 패턴에 적응하여 높은 성능을 유지할 수 있으며, 또한 자주 갱신되는 데이터와 그렇지 않은 데이터가 뒤섞여 쓰레기 수집 비용이 지속적으로 높아지는 페이지 사상 기법의 문제점을 해결할 수 있다.

본 논문에서 제안하는 Janus-FTL에서 논리적인 섹터들은 블록 사상 영역이나 페이지 사상 영역 중 한 곳에 존재할 수 있다. Janus-FTL은 섹터가 저장 된 사상 영역을 변경하는 두 가지 연산을 제공하는데, 하나는 블록 사상으로 관리되는 데이터 블록을 페이지 사상 영역에 편입시키는 퓨전(Fusion) 연산이고, 다른 하나는 페이지 사상 영역으로 관리되는 데이터 블록 하나를 블록 사상 영역으로 이동시키는 디퓨전(Defusion) 연산이다. 이 두 가지 연산을 사용하여 Janus-FTL은 페이지 사상 영역에 포함된 블록 개수를 동적으로 조절할 수 있으며, 따라서 참조 패턴에 적응하면서 페이지 사상 영역의 쓰레기 수집 비용을 최적으로 유지하는 것이 가능하다. 또한 효율적으로 블록 사상 영역과 페이지 사상 영역의 크기를 조절하는 방법을 통해 Janus-FTL을 구현하였으며, 트레이스 기반 시뮬레이터를 통해 성능을 측정하였다.

논문의 구성은 다음과 같다. 다음 절에서는 플래시 메모리의 특징을 설명하고, 플래시 메모리 기반 저장 장치와 관련된 연구를 제시한다. 3절은 Janus-FTL의 전체적인 구조와 퓨전 및 디퓨전 연산을 설명한다. 4절에서 다양한 트레이스를 이용하여 측정한 실험 결과를 설명하고, 마지막으로 5절에서 본 논문의 결론을 맺는다.

## 2. 관련 연구

플래시 메모리의 특정 페이지에 저장된 데이터를 갱신하기 위해서는 페이지가 위치한 블록을 소거한 후 블록 내 모든 페이지를 기존 데이터 또는 새로운 데이터로 다시 기록해야 한다. 그런데 이러한 갱신 기법은 비용이 매우 크기 때문에 실용적이지 않으며, FTL은 새로운 위치에 데이터를 기록하고 맵을 갱신하는 방법을 사용하여 데이터 갱신 비용을 줄인다. FTL이 사상하는 단위에 따라 크게 블록 사상 기법과 페이지 사상 기법으로 구분하고, 블록 사상 기법을 사용하면서 일부 여유 공간 또는 로그 영역을 페이지 사상 기법으로 관리하는 기법을 하이브리드 사상 기법이라 부른다.

페이지 사상 기법을 사용하는 플래시 메모리 저장 장치는 블록 사상 기법을 사용하는 것보다 순차 참조를 처리할 때 비효율적이다. 블록 사상 기법을 사용할 때 순차 참조는 결국 교환 병합(Switch merge)을 이용하여 플래시 메모리 블록을 재활용하는데, 대부분의 구현에서 이 연산은 페이지 사상 FTL의 쓰레기 수집 연산의 최소 비용보다 같거나 더 작다. 또한 페이지 사상 FTL의 경우 시간이 지남에 따라 전체적인 성능이 저하되는 현상이 발생한다. 이 현상은 점차 자주 참조되는 데이터와 그렇지 못한 데이터가 플래시 메모리의 블록에 뒤섞이면

서 발생하며, 이 경우 쓰레기 수집 비용이 점차 증가하면서 플래시 메모리 저장 장치의 성능이 저하된다.

이와 같이 블록 사상 기법과 페이지 사상 기법은 장단점을 가지며, 이들의 단점을 극복하고 장점을 수용하기 위하여 다양한 기법들이 제안되었는데, 이들의 목표는 Janus-FTL의 목표와 유사하다. Lee 등은 전체적으로 블록 사상 기법을 사용하면서 로그 블록 영역에서만 페이지 사상 기법을 사용하는 FAST(Fully Associated Sector Translation) 기법을 제안하였다[4]. 또 다른 연구로 [6]에서는 블록 사상 FTL에서 이주 연산과 기존 병합 연산을 혼합하여 반복 쓰기의 성능을 최적화 하는 기법을 제안하였다. Kim 등은 블록 사상 FTL을 사용하는 SSD의 성능을 향상하기 위하여 블록 내에서 비 연속적인 섹터들이 기록될 때 중간에 빠진 섹터들을 미리 복사하는 "페이지 패딩" 기법을 제안하였다[7].

Janus-FTL은 페이지 사상 기법과 블록 사상 기법을 같이 사용한다는 점에서 하이브리드 기법들과 유사하다. 그렇지만 Janus-FTL에서는 섹터가 포함된 블록 자체가 페이지 사상 영역 또는 블록 사상 영역으로 동적으로 이동이 가능하며, 페이지 사상 영역의 블록들은 명시적인 제한 없이 다른 블록의 섹터 데이터를 포함하거나 쓰레기 수집에 참여할 수 있다. 이러한 점들은 로그 블록과 같은 일부 영역에 대해서만 페이지 사상 기법을 사용하는 하이브리드 기법들과 다르다. 특히 Janus-FTL은 블록이 페이지 사상 영역 또는 블록 사상 영역으로 이동하는 퓨전과 디퓨전 연산을 가지며, 참조 패턴에 따라 Janus-FTL이 동적으로 각 영역의 크기를 조절할 수 있는 것이 기존 기법들과 다른 점이다.

## 3. Janus-FTL

한정된 공간을 이용하여 쓰레기 수집을 수행해야 하는 페이지 사상 기법의 성능을 개선하고, 순차 쓰기와 임의 쓰기 모두에게 최적의 저장 방식을 제공하기 위하여 제안된 Janus-FTL은 순차 쓰기와 임의 쓰기가 혼합된 작업 부하에 적합하도록 설계되었다. 이를 위해 Janus-FTL은 일부 데이터는 순차적으로 블록 사상 영역에 저장하고, 일부 데이터는 비 순차적으로 페이지 사상 영역에 저장한다.

Janus-FTL의 기본 동작은 최근에 비 순차적으로 쓰기가 요청된 데이터는 페이지 사상 영역에 저장하고, 나머지 데이터들은 순차적으로 블록 사상 영역에 저장한다. 즉, Janus-FTL은 대부분의 섹터들을 블록 사상 기법과 유사하게 특정 논리 블록에 순차적으로 저장된다. 앞으로 이렇게 섹터들을 순차적으로 저장하고 있는 블록을 "순차 블록"이라 부르고, 이와 같이 순차 블록들을 대상으로 블록 사상 기법을 적용하는 공간을 "순차 영역"이라 부른다. 이와는 다르게 최근에 비 순차적으로 쓰기가 요청된 섹터가 소속된 블록은 "퓨전 영역"이라는 불리는, 서로 다른 논리 블록에 속하는 섹터들이 혼합된 형태로 저장되는 영역에 소속된다. 이렇게 퓨전 영역에 소속된 블록은 "퓨전 블록"이라 부른다. 블록은 그 참조 지역성에 따라 퓨전 영역에 소속되었다가 다시 순차 영역으로 되돌아 올 수 있다. 그 이외에 Janus-FTL은 약간의 여유 블록을 가지며, 이들을 이용하여 퓨전 영역에 속한 블록들을 쓰레기 수집 형태로 재활용한다. 여유 블록의 수는 FTL의 설정에 따라 미리 정해진다. 만약 여유 블록이 n개이고 퓨전 영역에 포함된 논리 블록의 수가 m이라면, 퓨전 영역의 이용율은 m/(n+m)이며, 따라서 퓨전 영역에 포함된 논리 블록의 수가 많아질수록 퓨전 영역의 이용율은 증가하며 쓰레기 수집 비용 역시 증가한다.

Janus-FTL의 기본적인 동작 방법은 다음과 같다. 비 순차적인 쓰기 요청은 퓨전 영역에서 처리하고 이 영역의 크기는 비 순차적 쓰기 요청의 지역성과 양에 따라 크기가 변화한다. 퓨전 영역에서 미사용 페이자가 없는 경우 쓰레기 수집을 통해 여유 블록을 생성하며, 이 블록을 소거한 후 이곳의 페이지들에 새로운 데이터를 기록한다. 또한 퓨전 영역에 소속된 블록이 더 이상 적극적으로 사용(기록)되지 않으면 디퓨전 연산을 통해 순차 영역으로 이동시킨다.

그림 1은 간단한 구성을 예로 들어 퓨전과 쓰레기 수집 과정을 보여주며, 퓨전 영역은 세 개의 블록을 여유 공간을 가지도록 설정되었다. 그림 1(a)는 현재 어떠한 블록도 퓨전되지 않았으며 Janus-FTL의 초기 상태라 할 수 있다. 이때 17번 블록의 3번 페이지(그림에서는 17:3로 표현)에 기록된 데이터에 쓰기 요청이 발생한다면, 그림 1(b)와 같이 17번 블록이 퓨전 영역으로 포함된다. 그리고 변경된 데이터는 퓨전 블록 내의 미사용 페이지(2번 퓨전 블록의 첫 번째 페이지)에 기록된다. 다음으로 12번 블록의 1번 페이지에 기록된 데이터가 변경되는 경우 12번 블록 역시 퓨전 영역에 포함되며, 이곳에 포함된 데이터들이 변경되면 퓨전 영역의 미사용 페이지에 기록된다(그림 1(c)). 그림 1(c)에서 12번 블록의 1번과 4번 페이지의 데이터가 변경되어 2번 블록의 페이지들에 기록되었으며, 17번 블록의 3번, 0번, 1번 페이지들의 데이터가 변경되어 2번 블록의 페이지들에 기록되었다.

Janus-FTL은 쓰레기 수집과 디퓨전을 위해 항상 2개 이상의 여유 블록을 유지한다. 그림 1(c)와 같이 만약 여유 블록이 두 개밖에 없고 다른 블록에 미사용 페이지가 없으면, 새로운 쓰기 요청을 처리하기 위해 Janus-FTL은 쓰레기 수집을 수행해야 한다. 먼저 빈 블록 하나를 선택하고 소거한다(그림 1(d)에서 1번 블록). 그리고 그림 1(d)와 같이 쓰레기 수집 대상으로 3번 블록

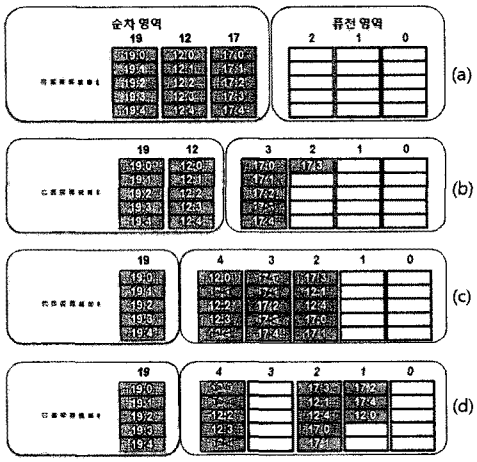



그림 1 퓨전과 쓰레기 수집 과정

을 선택하면, 이 곳의 유효 데이터를 방금 소거한 1번 블록으로 복사한다. 복사가 끝나면 이제 0과 3번 블록이 미사용 블록이 된다. 그리고 퓨전 영역에 소속된 데이터에 대한 쓰기 요청은 1번 블록의 미사용 페이지에 기록된다.

그림 2는 퓨전 영역에 포함된 한 무리의 데이터들이 (이해를 돕기 위해 예전에 17번 블록에 소속되었던 데이터들로 가정) 더 이상 빈번하게 변경되지 않는다고 판단되는 경우, 디퓨전 연산으로 이들을 순차 영역으로 이동시켜 퓨전 영역의 이용율을 낮추는 과정을 보여준다. 먼저 여유 블록을 소거한다. 그리고 예전에 17번 블록에 속하던 모든 데이터를 여유 블록으로 복사하고(그림 2(b)), 이 여유 블록을 순차 영역으로 포함시켜 블록 사상 기법으로 관리 하도록 한다. 이 결과로 퓨전 영역에 소속된 블록의 개수가 감소하며, 퓨전 영역의 이용율 역시 감소한다.

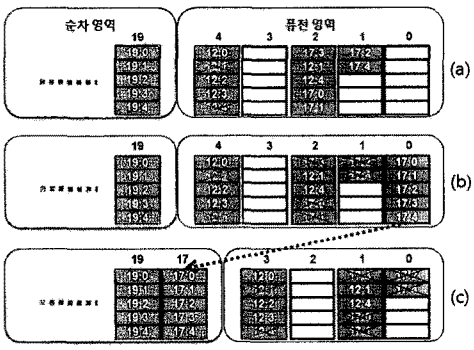



그림 2 디퓨전 과정

## 4. 실험 및 결과 분석

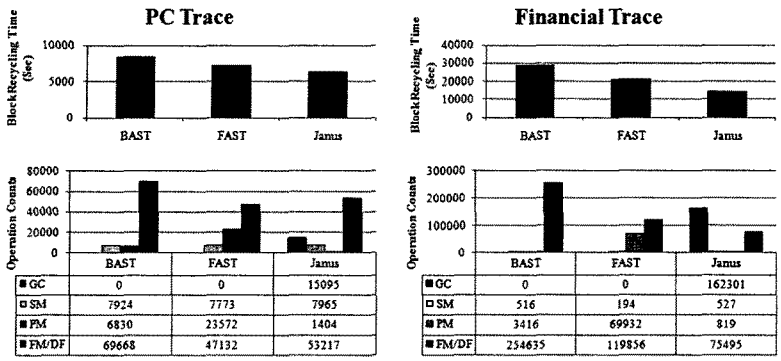
실험은 트레이스 기반 시뮬레이터에서 수행하였으며, 하이브리드 사상 기법을 사용한 BAST[3]와 FAST[4], 그리고 Janus-FTL을 비교했다. 시뮬레이터의 플래시 메모리는 [8]을 기반으로 구성하였으며, 저장장치의 공간은 1GB이다. 실험에 사용된 트레이스는 [5]의 PC Trace와 [9]의 OLTP Financial 트레이스를 이용하였다. PC Trace는 윈도우 XP기반 노트북 PC와 데스크탑 PC에서 문서 편집기, 음악 재생기, 웹 브라우저, 게임과 같은 다양한 응용 프로그램을 수행할 때 나타나는 저장장치로의 접근을 추출한 작업부하다. 그리고 OLTP Financial 트레이스는 금융기관의 OLTP 응용프로그램에서 추출된 것으로 비 순차적인 많은 쓰기를 발생하는 특징을 가진 작업부하다.

모두 FTL에서 동일한 비교를 위해 여유 블록의 개수는 전체 플래시 메모리 블록(4096개)의 2%(81개)를 사용한다. 실험결과에서 실제 수행 시간에 영향을 미치는 것은 블록 재활용에 걸리는 시간이며, 그 시간은 [9]에 나타난 시간을 기준으로 계산된다.

그림 3은 각각 트레이스를 수행했을 때 블록 재활용에 걸린 시간이며 아래의 그래프는 블록 재활용에서 쓰레기 수집(GC), 교환 병합(SM), 부분 병합(PM), 전체 병합(FM) 또는 디퓨전(DF)가 수행된 횟수이다. 기본적으로 모든 실험에서 Janus-FTL이 다른 FTL에 비해 블록 재활용에 드는 읽기, 쓰기, 소거 연산의 횟수가 적었으며, 그 결과 블록 재활용 시간도 가장 빠르게 나타났다.

PC Trace 실험의 경우 세 가지 정책 모두에서 교환 병합의 개수는 비슷하다. 이것은 모두가 순차적으로 발생하는 쓰기에 대해서는 최적으로 처리하고 있음을 보여준다. BAST가 FAST에 비해 성능이 떨어지는 것은 블록 스레싱(Block-Thrashing)[4] 현상으로 인한 것이며, FAST는 로그 블록을 페이지 사상 기법을 활용하여 이와 같은 문제를 해결했다. Janus-FTL은 디퓨전 횟수가 다른 FTL의 전체 병합 연산보다 많다. 하지만 디퓨전 연산은 퓨전 영역의 이용율에 의해 비용이 결정되므로 전체 병합의 비용보다 작을 수도 있고 클 수도 있다. 결국 적절한 디퓨전이 퓨전 영역의 활용도를 높이게 되어 전체 성능에서 좋은 결과를 보였다.

Financial은 순차적인 쓰기가 거의 존재하지 않는다. 따라서 모든 FTL에서 교환 병합 연산도 거의 없다. 위의 결과와 마찬가지로 BAST는 블록 스레싱 현상으로 인해 많은 전체 병합이 나타난다. FAST가 BAST에 비해 부분 병합이 많이 나타나는데 이것이 성능 향상에 영향을 미치는 것은 아니다(PC Trace 실험도 마찬가지다.). FAST는 0번 페이지에 해당하는 쓰기 요청이 발생했을 때 그것을 순차 쓰기로 판단한다. 하지만 Financial의 경우 순차적이면서 연속적인 쓰기가 거의 없기 때문에 또 다른 블록의 0번에 해당하는 쓰기가 요청되었을 경우 FAST는 나머지 모든 페이지를 복사하는 부분 병합 연산




| | BAST | FAST | Janus |
|---|---|---|---|
| GC | 0 | 0 | 15095 |
| SM | 7924 | 7773 | 7965 |
| PM | 6830 | 23572 | 1404 |
| FM/DF | 69668 | 47132 | 53217 |

| | BAST | FAST | Janus |
|---|---|---|---|
| GC | 0 | 0 | 162301 |
| SM | 516 | 194 | 527 |
| PM | 3416 | 69932 | 819 |
| FM/DF | 254635 | 119856 | 75495 |

그림 3 블록 재활용 비용과 연산의 수행 횟수

을 수행한다. 이렇게 동작하는 부분 병합 연산은 성능에 오히려 좋지 않은 영향을 미친다. 하지만 BAST에서 발생하는 블록 스레싱 현상이 더욱 큰 영향을 미치기 때문에 성능에서는 FAST가 앞섰다. 그렇지만 Janus-FTL의 경우 순차 영역에 할당 받았더라도 일정 개수이상 페이지가 사용되지 않으면 퓨전 영역으로 옮겨서 나머지 여유 페이지를 적극적으로 활용하게 되므로 더욱 좋은 결과를 보일 수 있다. 또한 위의 결과에서는 적절한 디퓨전과 쓰레기 수집으로 FAST보다 더욱 좋은 성능을 보였다.

## 5. 결론 및 향후 연구

본 논문에서는 블록 단위 사상과 페이지 단위 사상 방식을 선택적으로 사용하면서, 블록 사상 영역과 페이지 사상 영역의 크기를 참조 패턴에 따라 동적으로 변화시키는 Janus-FTL과 각 사상 영역들로 블록을 이동시키는 퓨전/디퓨전 연산을 제안하였다. Janus-FTL은 퓨전을 통해 자주 참조되는 데이터들의 갱신 비용을 적극적으로 낮추고, 반대로 자주 참조되지 않는 블록을 대상으로 디퓨전을 수행하여 퓨전 영역의 이용율을 적절한 수준으로 유지함으로써 쓰레기 수집 비용을 줄인다. 그리고 실험 결과에 따르면 Janus-FTL이 기존의 하이브리드 기법 FTL에 비해 최대 50%의 성능 향상이 있었다.

하지만 블록 사상 기법의 영역과 페이지 사상 기법의 영역의 크기를 간단하게 참조 패턴만을 고려하여 조절하였다. 따라서, 향후 연구에서는 먼저 각 사상 기법에 따른 블록 재활용 기법의 비용 모델 통해 두 가지 기법에 해당하는 영역의 최적의 크기를 결정하는 수학적인 모델을 세우고 이 모델의 검증이 필요하다.

## 참 고 문 헌

[1] Understanding the Flash Translation Layer (FTL) Specification: Intel Corporation, 1998.

[2] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Transactions on Computer Systems*, vol.10, no.1, pp.26-52, 1992.

[3] J. Kim, J. M. Kim, S. H. Noh, S. L. Min and Y. Cho, "A Space-efficient Flash Translation Layer for CompactFlash Systems," *IEEE Transactions on Consumer Electronics*, vol.28, no.2, pp.366-375, 2002.

[4] S. W. Lee, D. J. Park, T. S. Chung, D. H. Lee, S. Park and H. J. Song, "A Log Buffer based Flash Translation Layer using Fully Associative Sector Translation," *ACM Transactions on Embedded Computing Systems*, vol.6, no.1, Feb., 2007.

[5] S. Lee, D. Shin, Y.-J. Kim and J. Kim, "LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems," *ACM SIGOPS Operating. Systems Review*, vol.42, no.6, pp.36-42, 2008.

[6] J. Lee, S. Kim, H. Kwon, C. Hyun, S. Ahn, J. Choi, D. Lee and S. H. Noh, "Block Recycling Schemes and Their Cost-based Optimization in NAND Flash Memory based Storage System," in *Proceedings of 7th ACM & IEEE International Conference on Embedded Software*, pp.174-182, 2007.

[7] H. Kim, J. H. Kim, S. Choi et al., "A Page Padding Method for Fragmented Flash Storage," Lecture Notes in Computer Science, vol. 4705, pp. 164-177, 2007.

[8] "1G x 8Bit / 2G × 8Bit NAND Flash memory (K9L8G08U0M) Data Sheets," Samsung Electronics, Co., 2005.

[9] A. Gupta, Y. Kim and B. Urgaonkar, "DFTL: A Flash Translation Layer Employing Demand-Based Selective Caching of Page-Level Address Mappings," in *Proceeding of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.229-240, 2009.